Mälardalen University Doctoral Thesis
No.292

Automated Approaches for Formal Verification of Embedded Systems Artifacts

Predrag Filipovikj

June 2019

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

ISSN 1651-4238
ISBN 978-91-7485-429-9
Printed by E-Print AB, Stockholm, Sweden
Distribution: Mälardalen University Press

Abstract

Modern embedded software is so large and complex that creating the necessary artifacts, including system requirements specifications and design-time models, as well as assuring their correctness have become difficult to manage. One challenge stems from the high number and intricacy of system requirements that combine functional and possibly timing or other types of constraints, which make them hard to analyze. Another challenge is the quality assurance of various design-time models developed using Simulink as the *de facto* standard model-based development tool in the automotive domain, avionics domain, etc. Currently, the industrial state-of-practice resorts to simulation of Simulink models, which gives insight in the system's behavior yet does not provide a high degree of assurance that the model behaves correctly. A potential way to address the aforementioned challenges is to apply computer-aided, mathematically-rigorous methods for specification, analysis and verification already at the requirements specification stage, but also at later development stages.

In this thesis, we propose a set of approaches for the formal specification, analysis and verification of system requirement specifications and design-time Simulink models, with particular focus on the automotive industry. Our contributions are as follows: first, we assess the expressiveness of an existing pattern-based technique for the formal requirements specification on an operational system. Based on the positive findings, we deem the technique expressive enough to capture systems requirements in controlled natural language, from which formal counterparts can be automatically generated. To bring the approach closer to the practitioners we propose a tool, called ProPas. Next, we propose an automated consistency analysis approach based on Satisfiability Modulo Theories for the system requirements specifications formally encoded as temporal logic formulas. The approach is implemented in our ProPas tool

and is suitable to analyze the lack of logical contradictions within the system specification, at early system development phases. Our next contribution addresses the formal analysis and verification of large Simulink models. First, we propose a pattern-based and execution-order-preserving approach for transforming Simulink models into networks of stochastic timed automata, which can be analyzed using the UPPAAL SMC tool that returns the probability that a property is satisfied by the model. For the automated generation of the analysis model, we propose the SIMPPAAL tool. Our second approach is based on bounded model checking and is suitable for checking invariance properties of Simulink models. Compared to the statistical model checking approach, the invariance checking is reduced to a satisfiability problem. In case of property violation, the procedure generates a counter-example execution trace, which can be used for refining the model. In the same work we show that there exist commonly-used design patterns in Simulink models, for which the verification result is complete. The approach is supported by our SYMC tool.

For validation of the specification patterns, and the PROPAS tool we perform a case-study evaluation with practitioners, in collaboration with our industrial partner Scania. The results show that the pattern-based approach and the PROPAS tool can be practically useful in industrial settings. We apply the statistical model-checking approach and the SIMPPAAL tool on two industrial use cases, namely Brake-by-Wire and Adjustable Speed Limiter from Volvo Group Trucks Technology, which yields encouraging results. Finally, we validate the bounded invariance-checking approach and the SYMC tool on the Brake-by-Wire system, where we demonstrate both complete and incomplete verification of invariance properties.

Sammanfattning

Modern inbyggd mjukvara är ofta så stor och komplex att det blivit svårt att skapa nödvändiga artefakter, inklusive systemkravspecifikationer och designmodeller, samt att säkerställa att de är korrekta. En utmaning kommer från det stora antalet komplicerade systemkrav som kombinerar funktionalitet med tidskrav eller andra typer av begränsningar, vilket gör dem svåra att analysera. En annan utmaning är kvalitetssäkringen av olika designmodeller som utvecklats med Simulink, ett modellbaserat utvecklingsverktyg som är de facto-standard inom bland annat fordons- och flygindustrin. För närvarande förlitar sig industrin till stor del på simulering av Simulink-modeller, vilket ger insikt i systemets beteende men inte någon hög grad av försäkran att modellen beter sig korrekt. Ett möjligt sätt att ta itu med dessa utmaningar är att använda datorstödda, matematiskt rigorösa metoder för specifikation, analys och verifikation redan vid kravspecifikationen, men också under senare utvecklingsstadier.

I denna avhandling föreslår vi en uppsättning metoder för formell specifikation, analys och verifikation av systemkravspecifikationer och Simulinkmodeller, med särskild inriktning på bilindustrin. Våra bidrag är följande: För det första bedömer vi uttryckskraften hos en befintlig mönsterbaserad teknik för formell kravspecifikation på ett operativsystem. Baserat på de positiva resultaten bedömer vi att tekniken är tillräckligt uttrycksfull för att fånga systemkrav i kontrollerat naturligt språk, från vilka formella motsvarigheter kan genereras automatiskt. För att få tillvägagångssättet närmare utövarna har vi utvecklat verktyget ProPas. Därefter föreslår vi en automatiserad analysmetoder för konsistens, baserat på Satisfiability Modulo Theories, för systemkravspecifikationer formellt kodade som temporala logikformler. Tillvägagångssättet implementeras i vårt ProPas-verktyg och är lämpligt för att analysera bristen på logiska motsättningar inom systemspecifikationer under tidiga systemutvecklingsfaser. Vårt nästa bidrag rör formell analys och ver-

ifiering av stora Simulink-modeller. För det första föreslår vi ett mönsterbaserat och exekveringsordningsbevarande sätt att omvandla Simulink-modeller till nätverk av stokastisk tidsautomater, som kan analyseras med hjälp av UPPAAL SMC-verktyget som returnerar sannolikheten att modellen uppfyller en viss egenskap. För den automatiska genereringen av analysmodellen har vi utvecklat SIMPPAAL-verktyget. Vårt andra tillvägagångssätt är baserat på begränsad modellkontroll och är lämplig för att kontrollera invariansegenskaper hos Simulink-modeller. Jämfört med den statistiska kontrollmetoden reduceras invariantkontrollen till ett satisfierbarhetsproblem. Om egenskapen inte är uppfylld genererar metoden ett motexempel som kan användas för att förbättra modellen. I samma arbete visar vi att det finns vanligt förekommande designmönster i Simulink-modeller, för vilka verifieringsresultatet är fullständigt. Tillvägagångssättet stöds av vårt SYMC-verktyg.

För validering av specifikationsmönstren och PROPAS-verktyget utför vi en fallstudieutvärdering i samarbete med vår industriella partner Scania. Resultaten visar att det mönsterbaserade tillvägagångssättet och PROPAS-verktyget kan vara praktiskt användbara i industrin. Vi tillämpar den statistiska modellkontrollmetoden och SIMPPAAL-verktyget på två industriella användningsfall, nämligen Brake-by-Wire och Adjustable Speed Limiter från Volvo Group Trucks Technology, med goda resultat. Slutligen validerar vi den begränsade invariantkontrollmetoden och SYMC-verktyget på Brake-by-Wire-systemet, där vi demonstrerar både fullständig och ofullständig verifiering av invariansegenskaper.

To my parents

"Leave it be
It was meant for me
Soul sacrifice
Forgot the advice

Lost track of time
In a flurry of smoke
Waiting anxiety
For a fair judgement deserved"

A Fair Judgement, ***Opeth***

Acknowledgements

"Mom, you know, if I ever decide to move abroad, I would only consider moving to Sweden." - sixteen year old me, with absolutely no idea or vision about future. Around fifteen years down the road, here I am, sitting in my office writing the acknowledgements for my doctoral thesis. In Sweden.

My time as a PhD student has been everything but a smooth sail. If I think more, it can be probably best described as a roller-coaster ride full of glorious ups and downfalls of epic proportions. Luckily, I was never alone. My parents were always there for me, without excuses. No matter what. Always there! Mom, Dad, thank you for everything that you have done for me. Without you I would not have made it. You were always there to believe in me when no one else did. I am very sorry that I had to leave home to pursue my dream, but I dared to go this far only because I knew that you will always have my back. I dedicate all of my current and future achievements to you, because without you I am nothing!

First, I would like to thank my advisors. The biggest token of appreciation goes to my main advisor, Associate Professor Cristina Seceleanu. Thank you for giving me the opportunity to become a PhD student. You have been an excellent advisor who have taught me many invaluable lessons, both in research and in life in general. This thesis, at least in this shape and form, would not have been possible without you. Many thanks to my co-advisor, Dr. Guillermo Rodriguez-Navas for the collaboration that we had during the years. I learned a lot from you, especially how to stay positive in the darkest of times. I would like to thank my industrial co-advisor Professor Mattias Nyberg. You always had faith in me and my work, and for that I am very grateful. Last but not least, I would like to thank Professor Hans Hansson. Even though you were my main advisor for only one year, it was a pleasure and honor to be your student.

Pursuing a PhD within a project that is an academic-industrial cooperation

is an amazing but also challenging experience. I would like to thank all the people from our industrial partners, Scania AB CV and Volvo Group Trucks Technology who were involved in the VeriSpec project in one way or another. Special thanks goes to Oscar Ljungkrantz and Henrik Lönn from Volvo and Jon Andersson from Scania. I will never forget the interview with Jon at the Scania Technology Center in the cabin of a test truck.

I would like to express my gratitude to the faculty examiner, Professor Jim Woodcock and the grading committee members: Professor Kim Larsen, Associate Professor Luigia Petre, and Associate Professor Christian Berger, for kindly accepting our invitation and dedicating part of their valuable time to review my work. I am truly honored to have you as the committee who validates my work.

I would also like to thank Associate Professor Alessandro Papadopoulos for providing feedback on an earlier version of the thesis, and Professor Jan Carlson for helping with the Swedish version of the abstract.

During the fall of 2017, I had the privilege to visit the Chair for Software Modeling and Verification (MOVES) at Aachen University in Germany led by Professor Joost-Pieter Katoen where I had the opportunity to be a part of an amazing group of researchers. When I look back, it was probably one of the most inspiring times of my doctoral studies. Thank you Joost-Pieter for accepting me as a guest researcher, and to all of the students, researchers and chair stuff members for the wonderful treatment while I was there. You made Aachen and MOVES feel like home.

Research is only one aspect of the graduate education in Sweden. Taking courses (quite a few of them!) and teaching are two other aspects that play crucial role in the education and development of a doctoral student in Sweden. I am using this opportunity to thank all the incredible professors and lectures at the university whose courses I had the pleasure to take. Regarding my teaching duties, I had the privilege to be a teaching assistant for three amazing teachers from the department: Professor Ivica Crnkovic, Professor Jan Carlson and Dr. Severine Sentilles. It was a great pleasure to work with each one of you. I would like to emphasize that Ivica is the main reason why I chose academia. I have always admired your work ethics, professionalism, and above all how nice human being you are. You are such an inspiration and a role model!

The IDT department at Mälardalen Univerisity, where I have spent most of the past five years of my life is an amazing place. It is not amazing because of the fancy offices equipped with perfect air-conditioning and heating, but because of the people who work there. I would like to express my deepest appreciation to all of the senior research and academic staff, the ladies from

the administrative department and especially the fellow PhD students for making the time spent at the department joyful. My special shout-out goes to the espresso gang: Alessandro, Matthias (now at KTH), Mirgita, and Saad. The coffee trips to the espresso machine located at the Software Engineering division have always been the highlights of my working days. Of course, that would not have been possible without Radu Dobrin, the leader of the Software Engineering division, who was courageous enough to buy espresso machines so that people can enjoy a nice cup of coffee. Thank you Radu, from the bottom of my heart! Raluca and Eddie, you were not only a great colleagues and coauthors, but also great friends. Thanks for all the fun times we had together. Finally, I would like to specially thank my academic sister Aida for all the nice things that she has done for me during the past six years. Our relationship has gone through many phases, starting from you being my master's thesis supervisor, then a colleague, and eventually becoming a dear friend. I always felt you truly cared for me, and for that I am forever indebted to you.

There are some people personally close to me that throughout the years have influenced me in many ways. To my cousins, Emilija and Zoran. I never considered you as cousins, but rather as my siblings. Despite being a single child you never let me feel alone, and you always took good care of me. Thanks for always being there, no matter the circumstances. Another very influential person in my life was my grandmother Gorka, who always wanted me to become an engineer. Sadly, she passed away before I became one. Being where I am now, there is some inexplicable joy and satisfaction in the fact that I did not let her down.

To Dragana, Dimitar, and Jani. Life was a bit unfair when we scattered around the world, but despite the distance, we managed to keep the friendship alive. You are truly special people, and I feel very privileged to have you as my friends.

Predrag Filipovikj
Västerås, April, 2019

List of publications

Publications Included in the Thesis[1]

Paper A *Reassessing the Pattern-Based Approach for Formalizing Requirements in the Automotive Domain*. **Predrag Filipovikj**, Mattias Nyberg, Guillermo Rodriguez-Navas. In the Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE'14), pages 444-450. Karlskrona, Sweden. August, 2014. IEEE Computer Society.

Paper B *Automated SMT-based Consistency Analysis of Industrial Critical System Requirements*.[✢] **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Mattias Nyberg, Cristina Seceleanu. ACM SIGAPP Journal of Applied Computing Review, pages 15 – 27. Volume 17, Number 4. December, 2017. ACM.

[✢]This article is an extended version of the following conference paper: *SMT-based Consistency Analysis of Industrial Systems Requirements*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Mattias Nyberg, Cristina Seceleanu. In the Proceedings of the 32nd ACM Symposium On Applied Computing (SAC 2017), pages 1272 – 1279. **Best Paper Award**. Marrakesh, Morocco. April, 2017. ACM.

Paper C SIMPPAAL - *A Framework For Statistical Model Checking of Industrial Simulink Models*.[⋆] **Predrag Filipovikj**, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz , Henrik Lönn. ACM Journal of Transactions on Software Engineering and Methdology. Revisions required. Submitted in November, 2018.

[1]The included publications are reformatted to comply with the thesis printing format.

⋆This article is an extended version of the following conference paper: *Simulink to* UPPAAL *Statistical Model Checker: Analyzing Automotive Industrial Systems*. **Predrag Filipovikj**, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz, Henrik Lönn. In the Proceedings of the 21st International Symposium on Formal Methods (FM2016), pages 748–756. Limassol, Cyprus. November, 2016. Springer, LNCS.

Paper D *Bounded Invariance Checking of Simulink Models*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Cristina Seceleanu. In the Proceedings of the 34th ACM Symposium of Applied Computing (SAC'19), pages 2155–2164. Limassol, Cyprus. April, 2019. ACM.

Paper E *Specifying Industrial System Requirements using Specification Patterns: A Case Study of Evaluation with Practitioners*. **Predrag Filipovikj** and Cristina Seceleanu. In the Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019). Heraklion, Crete. May, 2019. SciTePress Digital Library (Science and Technology Publications, Lda).

Additional Publications not Included in the Thesis[2]

1. *Bounded Verification of Simulink Models*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Cristina Seceleanu. Mälardalen Real-Time Research Center, Mälardalen University. December, 2018.

2. *Model-Checking-based vs. SMT-based Consistency Analysis of Industrial Embedded Systems Requirements: Application and Experience*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Cristina Seceleanu. Journal of Electronic Communications of the EASST, Vol. 75. October, 2018.

3. *An Energy-aware Mutation Testing Framework for EAST-ADL Architectural Models*. Raluca Marinescu, **Predrag Filipovikj**, Eduard Paul Enoiu, Jonatan Larsson , Cristina Seceleanu. The 29th Nordic Workshop on Programming Theory (NWPT'17). October, 2017.

4. *Analyzing Industrial Simulink Models by Statistical Model Checking*. **Predrag Filipovikj**, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz , Henrik Lönn. MRTC Report, Mälardalen Real-Time Research Center, Mälardalen University. March, 2017.

5. *SMT-based Consistency Analysis of Industrial Systems Requirements*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Mattias Nyberg, Cristina Seceleanu. In the Proceedings of the 32nd ACM Symposium On Applied Computing (SAC 2017), pages 1272 – 1279. **Best Paper Award**. Marrakesh, Morocco. April, 2017. ACM.

6. *Increasing Embedded Systems Quality through Automated Specification and Analysis of Requirements and Behavioral Models*. Predrag Filipovikj. The 43rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM2017), Student Research Forum. **Best Student Research Proposal Award**. Limerick, Ireland. January, 2017. Springer, LNCS.

7. *Simulink to* UPPAAL *Statistical Model Checker: Analyzing Automotive Industrial Systems*. **Predrag Filipovikj**, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz, Henrik Lönn. In the

[2]The publications are listed in reverse chronological order.

Proceedings of the 21st International Symposium on Formal Methods (FM2016), , pages 748 – 756. Limassol, Cyprus. November, 2016. Springer, LNCS.

8. *Integrating Pattern-based Formal Requirements Specification in an Industrial Tool-chain*. **Predrag Filipovikj**, Trevor Jagerfield, Mattias Nyberg, Guillermo Rodriguez-Navas, Cristina Seceleanu. In the Proceedings of the 10th IEEE International Workshop on Quality Oriented Reuse of Software (QUORS'16), collocated with COMPSAC 2016, pages 167 – 173, Volume 2. Atlanta, USA. June, 2016. IEEE Computer Society.

Contents

I

Thesis

Chapter 1

Introduction

Using embedded software to perform highly complex functions has been enabled by the ever-increasing computational power and memory capacity of the embedded hardware. The automotive industry is one of the many industries that have been profoundly impacted by the rise of embedded systems. With the "*x-by-wire*" technology that was introduced almost three decades ago, the modern vehicles have come a long way to become software intensive systems [1,2], in which even the core features such as the engine control and management, braking, steering, etc., are implemented in software. The pinnacle of this trend is made by the advanced driver assistance systems intended to either assist the drivers or to completely autonomously operate vehicles in a safer and more efficient manner. The embedded hardware in vehicles consists of distributed embedded computers called electrical control units (ECU). A modern premium car runs several tens of millions lines of code distributed over more than 70 independent ECU [3].

The increase in size and complexity of automotive software functions impacts all the phases of system development and the produced artifacts, including the system's specification, design and architecture, as well as the integration and testing phases [4]. Moreover, many of these functions are classified as *safety critical* [5], meaning that their malfunction can result in damages to the environment or potentially endanger human lives. Consequently, in order to increase the safety of the vehicles, the new ISO26262 standard [6] for automotive safety highly recommends using rigorous verification techniques for establishing the correctness of automotive functions. In this thesis, we show how to apply techniques for the rigorous verification of embedded software,

with specific focus on the design-time artifacts created during the embedded software development, which include system specifications and behavioral system models, with particular focus on the automotive domain. We assume that the complete set of characteristics and functionalities of an embedded software are determined during the design phase, and once deployed into operation they cannot be changed [7, 8].

Motivation. The predominant way of specifying requirements of automotive embedded software is by using free-text natural language. The requirements are usually organized in requirements specification documents called *system specifications*, which are created and managed mostly by using general purpose text editing software or in some cases specialized tools such as IBM Rational Doors [9]. The main advantage of systems specifications specified in natural language is that they are easy to read and interpret by various stakeholders in the system's development process, provided that the stakeholders have sufficient domain knowledge. However, despite enabling high versatility, such a way of specifying requirements suffers from several drawbacks. The most obvious one is the potential ambiguity of the specifications, which stems from the inherent ambiguity of the natural language itself. Consequently, there might be situations of the same requirement being interpreted in various ways. As such, the natural language ambiguity might affect the quality of the system specifications negatively. One such quality attribute is the *consistency* of the system specification, that is, the lack of internal logical contradiction between requirements.

The current industrial state-of-practice relies on manual peer-review as the predominant technique for assessing the quality of system specifications [10]. In this case, the combination of ambiguity of the natural language and the sheer size of the specifications might potentially render the quality assurance through manual peer-review of requirements ineffective. Under such circumstances, possible inconsistencies (logical contradictions) could escape the reviewer's eye, especially in cases of large specifications. A promising way to prevent the ambiguity of system requirements specification and to improve their analyzability is to employ rigorous computer-aided analysis and verification of requirements, enabled by formal techniques and *specification patterns* for formalization [11, 12]. Even though the feasibility of formal techniques has been demonstrated on industrial systems [13, 14], their actual adoption in the industrial development of embedded systems is hindered by the difficulty of producing formal system specifications.

The *model-based development* (MBD) paradigm has started to gain mo-

mentum in the automotive industry as the models provide a good way of abstracting the problem and documenting the design. As such, MATLAB *Simulink* [15] is the *de facto* standard MBD tool in the domain. The Simulink-based development revolves around producing behavioral models, which are treated as executable system specifications that can also be used as an input into specialized commercial tools, such as *Simulink Coder* [16] to automatically generate the code that is deployed in vehicles. For assuring the correctness of the Simulink models, the current state-of-practice techniques rely predominantly on simulation, which gives insight in the system's behavior yet does not provide a high degree of assurance that the model behaves correctly.

For the structural analysis of Simulink models using automated and rigorous techniques, the state-of-practice tool is *Simulink Design Verifier* [17], with its scope and applicability being limited to detecting errors such as buffer overflow, division by zero, array access violations, etc. By exploring the literature and discussing with practitioners, we have come to the conclusion that rigorous quality assurance of the industrial-size Simulink models against high-level properties such as invariance, timeliness, reachability etc., is actually lacking.

Contributions. Given the above motivation and mentioned gaps, in this thesis we propose a set of industrially-attractive approaches for the formal specification, analysis and verification of system requirements, and design-time behavioral models of embedded systems. The contributions follow two lines of research: i) formal specification and analysis of system requirements specifications, and ii) formal analysis and verification of Simulink models. For clarity, we divide the contributions into four major parts, as follows. As the **first contribution**, we propose methods and tools for the formal system specification via specification patterns, and automated consistency analysis of requirements modeled in temporal logic. For that purpose, we first assess the suitability and expressiveness of specification patterns [11, 12, 18] for formalizing system requirements in the automotive domain [19]. Our results show that the specification patterns are expressive enough to encode most of the requirements of complex automotive systems from Scania, Sweden. To make specification patterns attractive to industrial practitioners, we develop specialized tool support by proposing the SESAMM SPECIFIER tool [20], which is intended to aid the practitioners when using patterns. As the second part of the first contribution, we propose a consistency analysis approach based on *satisfiability modulo theories* (SMT) [21], intended to ensure that a system specification encoded as a set of temporal formulas is free of logical contradictions [22]. For full automation of our consistency analysis approach, we propose the PROPAS

tool [22], which is an extension of our SESAMM SPECIFIER tool that includes the SMTLIBREQ library [22] for the automated consistency analysis.

The **second contribution** of the thesis is an automated approach for the formal analysis of Simulink models based on *statistical model checking* [23], which uses the UPPAAL Statistical Model Checker [24] as the underlying analysis engine [25]. The core of our approach is a template-based and execution-order-preserving transformation of Simulink models into *networks of stochastic timed automata* [24]. To enable such a transformation, we first propose a semantic definition of Simulink blocks and their composition, as timed transition systems, based on the informal (execution) semantics documented by MathWorks, the vendor of the MATLAB Simulink tool. Our approach is suitable for analysis of both discrete-time and hybrid (models composed of discrete- and continuous-time blocks) Simulink models. For automation, we propose the SIMPPAAL tool [25] that generates the resulting formal model automatically. Since statistical model checking does not perform an exhaustive exploration of the model's state space, but instead checks the property based on a number of simulations of the model, it does not encounter the infamous state-space-explosion and thus has the potential to scale with the size and complexity of industrial Simulink models. Despite the fact that the approach is not exhaustive, it provides rigorous probabilistic guarantees of the correctness of a given Simulink model, with respect to invariance, reachability and certain liveness properties.

In order to remove encoding the Simulink model into a stochastic formal framework, as the **third contribution**, we propose an alternative approach for the formal verification of Simulink models based on *bounded model checking* [26]. The main use-case for the approach is the analysis of an underlying Simulink model with respect to invariance properties. Due to the fact that in general only a portion of the reachable state-space of the model is explored, the approach is not exhaustive, however it provides a full guarantee that a given property is satisfied by the bounded, reachable state-space of the model while overcoming the state-space-explosion problem. Compared to the statistical model checking, in case of property violation, one can generate a counter-example that can be used for improving the model. Additionally, we show that there are certain Simulink designs for which the bounded invariance checking is complete, thus the verification represents a full guarantee that the model satisfies a given invariance property. The approach is automated by our SYMC tool [26] that we propose in this thesis.

As the **fourth and final contribution** that we also propose in this thesis, we carry out validation of each of the proposed approaches over system specifica-

tions and Simulink models provided by our industrial partners, namely Scania and Volvo Group Trucks Technology (VGTT), both from Sweden. To assess the practical usefulness of the PROPAS tool for the various industrial stakeholders of the embedded software development process, we perform a case study evaluation with practitioners in collaboration with Scania [27]. For validation of our consistency analysis approach, we apply the PROPAS tool on the system specification of an operational system from Scania [22]. We validate the SIMPPAAL and the statistical model checking approach on two industrial Simulink models, namely, of a Brake-by-Wire (BBW) system prototype, and of Adjustable Speed Limiter (ASL), both from VGTT [25], whereas the SYMC tool and the bounded model checking are validated on one Simulink model only, that is, the Simulink model of the BBW system from VGTT [26].

1.1 Thesis overview

The thesis is divided into two main parts. The first part is an overall summary of the thesis, organized as follows. In Chapter 2, we give an overview of the background concepts that are used in the thesis. In Chapter 3, we present the research methodology that we apply to conduct the presented research, followed by the definition of the research goals in Chapter 4. In Chapter 5, we present a compact and more technical description of the thesis contributions and show their mapping to the previously defined research goals. The overview of and comparison to related work is given in Chapter 6, followed by conclusions and directions for future work in Chapter 7 that concludes the first part of the thesis.

The second part is given as a collection of five publications that encompass all thesis contributions. The included papers are the following:

Paper A. *Reassessing the Pattern-Based Approach for Formalizing Requirements in the Automotive Domain*. **Predrag Filipovikj**, Mattias Nyberg, Guillermo Rodriguez-Navas. In Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE'14), pages 444-450. Karlskrona, Sweden, August, 2014, IEEE CS.

Abstract. The importance of using formal methods and techniques for verification of requirements in the automotive industry has been greatly emphasized with the introduction of the new ISO26262 standard for road vehicles functional safety. The lack of support for formal modeling of requirements still represents an obstacle for the adoption of the formal methods in industry. This

paper presents a case study that has been conducted in order to evaluate the difficulties inherent to the process of transforming the system requirements from their traditional written form into semi-formal notation. The case study focuses on a set of non-structured functional requirements for the Electrical and Electronic (E/E) systems inside heavy road vehicles, written in natural language, and reassesses the applicability of the extended Specification Pattern System (SPS) represented in a restricted English grammar. Correlating this experience with former studies, we observe that, as previously claimed, the concept of patterns is likely to be generally applicable for the automotive domain. Additionally, we have identified some potential difficulties in the transformation process, which were not reported by the previous studies and will be used as a basis for further research.

Contributions. I was the main driver of the paper. I have performed most of the activities related to the case study, including the requirements gathering and extraction, applying the patterns and drawing conclusions. I also wrote most of the paper. Guillermo Rodriguez-Navas and Mattias Nyberg participated in discussions and contributed with ideas and comments on the patterning process.

Paper B. *Automated SMT-based Consistency Analysis of Industrial Critical System Requirements*[+]. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Mattias Nyberg, Cristina Seceleanu. ACM SIGAPP Journal of Applied Computing Review, pages 15 – 27. Volume 17, Number 4. December, 2017. ACM.

[+]This article is an extended version of the following conference paper: *SMT-based Consistency Analysis of Industrial Systems Requirements*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Mattias Nyberg, Cristina Seceleanu. In the Proceedings of the 32nd ACM SIGAPP Symposium On Applied Computing (SAC 2017), pages 1272 – 1279. **Best Paper Award**. Marrakesh, Morocco. April, 2017. ACM.

Abstract. With the ever-increasing size, complexity and intricacy of system requirements specifications, it becomes difficult to ensure their correctness with respect to certain criteria such as consistency. Automated formal techniques for consistency checking of requirements, mostly by means of model checking, have been proposed in academia. Sometimes such techniques incur a high modeling cost or analysis time, or are not applicable. To address such problems, in this paper we propose an automated consistency analysis technique of requirements that are formalized based on patterns, and checked using state-of-the-art Satisfiability Modulo Theories solvers. Our method assumes several

transformation steps, from textual requirements to formal logic, and next into the format suited for the SMT tool. To automate such steps, we propose a tool, called PROPAS, that does not require any user intervention during the transformation and analysis phases, thus making the consistency analysis usable by non-expert practitioners. For validation, we apply our method on a set of timed computation tree logic requirements of an industrial automotive system called the Fuel Level Display.

Contributions. I was the main driver of the paper. I collected the requirements which were included in the case study. I also performed the requirements formalization and extracted the used patterns. I wrote the structured derivations for the patterns from TCTL into first-order-logic and their encoding in Z3. I developed the PROPAS tool which was used in the case study. I wrote the complete paper. Cristina Seceleanu and Guillermo Rodrigues-Navas contributed with useful comments for the structured derivations and the overall structure of the paper. Mattias Nyberg provided feedback on the formalization of the requirements from the FLD system.

Paper C. SIMPPAAL - *A Framework For Statistical Model Checking of Industrial Simulink Models*⋆. **Predrag Filipovikj**, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz, Henrik Lönn. Submitted to the ACM Transactions on Software Engineering and Methodology (TOSEM) Journal. Revisions required. Submitted in November, 2018.

⋆This article is an extended version of the following conference paper: *Simulink to* UPPAAL *Statistical Model Checker: Analyzing Automotive Industrial Systems*. **Predrag Filipovikj**, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz, Henrik Lönn. In the Proceedings of the 21st International Symposium on Formal Methods (FM2016), pages 748–756. Limassol, Cyprus. November, 2016. Springer, LNCS.

Abstract. Nowadays, electronic brains control dozens of functions in vehicles, like braking, cruising, etc. Model-based design approaches, in environments such as MATLAB Simulink, seem to help in addressing the ever-increasing need to enhance quality, and manage system complexity, by supporting functional design from a set of block libraries that can be simulated and analyzed for hidden errors, but also used for code generation. Since such implementations are at most as correct as the models that they are generated from, providing assurance that Simulink models fulfill given functional and timing re-

quirements is desirable. Exhaustive verification methods like model checking might easily encounter the state-space explosion problem for large Simulink descriptions. To tackle such problem, in this paper, we propose a pattern-based, execution-order preserving automatic transformation of atomic and composite Simulink blocks into stochastic timed automata that can be formally analyzed with UPPAAL Statistical Model Checker. The latter employs scalable yet statistical methods for reasoning, rather than exhaustive ones. To enable the formal analysis, we first define the formal syntax and semantics of Simulink blocks and their composition, and show that the proposed transformation is provably correct for a certain class of Simulink models. Our method is supported by the SIMPPAAL tool, which we introduce and apply on two industrial Simulink models, of a prototype called the Brake-by-Wire system, and of an operational Adjustable Speed Limiter system. This work enables the formal analysis of industrial Simulink models, by automatically generating their stochastic timed automata counterparts.

Contributions. I was the main driver of the paper and main contributor together with Nesredin Mahmud, with whom we developed the template-based transformation of Simulink blocks into a network of stochastic timed automata. In addition, I designed and implemented the SIMPPAAL tool as well as a subset of the plug-ins for generating the block routines. I wrote three complete sections and three additional subsections in the paper. Further on, I applied the SIMPPAAL tool on the Brake-by-Wire Simulink model to generate the network of stochastic timed automata. Raluca Marinescu validated the correctness of the generated Brake-by-Wire UPPAAL network of stochastic timed automata model and performed the SMC analysis of the model. Nesredin Mahmud developed a subset of the block routine plug-ins and co-wrote all the sections related to the ASL system. Cristina Seceleanu wrote the proof of correctness for the transformation and one additional section of the paper. She also provided useful comments. Guillermo Rodriguez-Navas wrote the related work section and provided useful comments for the rest of the paper. Oscar Ljungkrantz and Henrik Lönn provided valuable feedback both on the approach and the final version of the paper. Myself and Cristina Seceleanu were the main responsible for polishing the manuscript and submitting it for review.

Paper D. *Bounded Invariance Checking of Simulink Models*. **Predrag Filipovikj**, Guillermo Rodriguez-Navas, Cristina Seceleanu. In the Proceedings of the 34th ACM Symposium of Applied Computing (SAC'19), April 8-12,

2019, Limassol, Cyprus. ACM.

Abstract. Currently, Simulink models can be verified rigorously against design errors or statistical properties. In this paper, we show how Simulink models can be formally analyzed for invariance properties using bounded model-checking reduced to satisfiability modulo theories solving. In its basic form, the technique provides means for rigorous verification of an underlying model over bounded traces, however, in general the procedure is incomplete. We identify common Simulink block types and compositions by analyzing selected industrial models, and we show that for some of them the set of non-repeating states (reachability diameter) can be visited with a finite set of paths of finite length, yielding the verification complete. We complement our approach with a tool, called SYMC that automates the following: i) calculation of the reachability diameter size for some of the designs, ii) generation of finite (bounded) paths of the underlying Simulink model and their encoding into SMT-LIB format, and iii) checking invariance properties using the Z3 SMT solver. To show the applicability of our approach, we formally analyze the Simulink model of a prototype industrial system, namely the Brake-by-Wire system from Volvo Group Trucks Technology, Sweden.

Contributions. I was the main driver for this work. I proposed the approach and implemented the SYMC tool. I performed the analysis of the industrial use-case models to identify the commonly-used compositions of blocks. I proposed all the definitions, theorems, and the necessary proofs. I wrote the complete paper. Cristina Seceleanu and Guillermo Rodriguez-Navas contributed through useful discussions on the approach, mostly on the completeness of the bounded model checking for the identified compositions. Additionally, Cristina provided a valuable feedback on the final draft of the paper.

Paper E. *Specifying Industrial System Requirements using Specification Patterns: A Case Study of Evaluation with Practitioners*. **Predrag Filipovikj** and Cristina Seceleanu. In the proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019), May 4-5, 2019, Heraklion, Crete. SciTePress Digital Library (Science and Technology Publications, Lda).

Abstract. With the ever-increasing size and complexity of the industrial software systems there is an imperative need for an automated, systematic and

exhaustive verification of various software artifacts, such as system specifications, models, code, etc. A potential remedy for this need might lie in a pool of techniques for computer-aided verification of software related artifacts, including system specifications. The Achilles' heel of these techniques, and the main hinder for their wider adoption in industrial development process are the complexity and the specialized skill-set required for the formal encoding of specifications. To alleviate this problem, *Specification Patterns* that are based on the observation that the system specifications are framed within reoccurring solutions have been proposed. The approach has been shown to be expressive enough for capturing requirements in the automotive domain, however, there is a lack of empirical data that can be used to judge its practical usefulness. In this paper, we involve an existing specification-patterns-based tool, and propose a small-size evaluation of the approach with practitioners, on a case study conducted in cooperation with Scania, one of the world's leading manufacturers of heavy-load vehicles. Our results show that the specification patterns that are supported by an adequate tooling have the potential for adoption in industrial practice.

Contributions. I was the main driver for this work. I performed all phases of the case study, including the case study planning, setup, execution and reporting. During the case study setup phase, I selected the subset of requirements that are used in the case study and redesigned the PROPAS tool specifically for this study, including adjustments to the user interface and implementing the features such as the timer that automatically keeps track of the time spent per requirement. Then, I proposed the group of case study subjects and personally contacted each of them to ensure their participation in the study. I prepared the tutorial used by the case study subjects. During the case study execution phase, I met with each of them at their workspace at Scania in Södertälje, Sweden. Finally, I did the data analysis and drew the conclusions. I wrote the complete paper. Cristina Seceleanu contributed through useful discussions especially for the development of the survey, mostly on defining the questions and the set of possible options for the expressing the degree of satisfaction. Additionally, Cristina provided valuable feedback on the final draft of the paper.

Chapter 2

Background

In this chapter, we introduce the technical concepts that are used throughout the thesis. First, in Section 2.1, we present a brief overview of the model-based development paradigm. Next, in Section 2.2 we describe specification patterns for formal system specification, followed by an overview of some state-of-the-art techniques for consistency analysis of the formally encoded specifications, in Section 2.3. After that, in Section 2.4, we present the MATLAB Simulink environment. Finally, in Section 2.5, we give a short overview of the formal techniques for modeling and verification of embedded systems, which we employ in our work, namely satisfiability modulo theories, statistical model checking and bounded model checking.

2.1 Model-based Development

Managing complexity of artifacts is one of the fundamental problems in all engineering disciplines. According to Brooks [28], the complexity in systems engineering stems from at least two sources: i) the inherent complexity of the engineering problem itself, and ii) the complexity introduced by the tools and methods that are used for solving the problem. In the domain of software engineering, *model-based development* (MBD) has proven to be an effective paradigm for developing complex software solutions. The main idea of MBD is to facilitate system modeling through multiple abstractions, corresponding to the different systems development phases. Ideally, this should enable the seamless integration of design and analysis techniques and tools throughout

the system development.

The process of abstraction focuses on removing the irrelevant details from the system. By applying abstraction one creates a specific view of the system that contains all the essential parts relevant for a specific problem. The newly obtained abstract view of the system is then more tractable for various purposes, such as analysis and verification. An abstract version of a system as observed from a particular point of view is called *model*. By using MBD paradigm for software development, the goal is to raise the abstraction level of both the underlying problem and the proposed (software) solution, and thus shift the focus from coding to modeling activities. According to Selic [29], in order for any software model to be practically useful, it should at least exhibit the following characteristics:

- *Abstract* - the model should be abstract, which means that it should hide all the irrelevant details such that the important features stand out;
- *Accurate* - the model must represent the abstracted system faithfully. This means that any model must correctly reflect the properties of interest;
- *Understandable* - the model must convey the information of interest in a clear and unambiguous way;
- *Predictable* - the model should behave in the expected way.

A software model that has the aforementioned properties is practically useful for documenting both the problem and the solution. The accuracy and the predictability features of the models allows one to treat them as executable specifications. This is possible only if the language used for describing the model has well-defined semantics, which is the main prerequisite for employing specialized tools for generating the executable code from the model directly.

Because of the benefits that it provides, the MBD paradigm has become the “go-to” way for developing software in the automotive systems domain. The paradigm has become an enabler for the engineers who are not trained as software engineers to produce most of the embedded software that is running in modern vehicles. By using the MBD methods and tools, engineers who are skilled in various engineering disciplines, but not necessarily in software engineering, can abstract away the implementation of their solutions, which lets them model their solutions in easy and intuitive ways. One such tool for

modeling, simulation and code generation in industrial settings is MATLAB *Simulink*, which we introduce in details later in this chapter.

2.2 Specification Patterns

Writing formal properties for formal verification is a daunting task which requires high proficiency in logic. One way of enabling practitioners who are not experts in formal techniques to create formal system specifications is to provide them with methods and tools for a structured and reusable style of specifying requirements, where the structures have precise semantics that define the relationship between their syntax and the model of computation. Having the system specifications expressed using such structures facilitates the automatic generation of the formal systems' specifications.

To enable the industrial practitioners to write formal specifications, in our work we adopt the *specification patterns system* (SPS) approach [11, 18]. The approach is based on the assumption that systems' specifications are framed within reoccurring solutions, from which a set of patterns can be extracted and saved for future reuse. Each pattern captures some system *behavior* which must hold for a certain extent of execution called *scope*. The patterns are expressed as a combination of literal and non-literal terminals.

The original SPS catalog proposed by Dwyer et al. [11, 18] is compiled by analyzing more than 500 examples of property specifications for various systems. The catalog contains 13 qualitative patterns, which for easier navigation are divided into two categories: *order* and *occurrence*, expressed in different temporal logics. The occurrence category contains patterns that describe the occurrence of a given state/event in the system, while the patterns from the ordering category are used to capture the relative ordering of the occurrence of multiple events/states during system execution. The catalog also introduced five different scopes, given as following:

- *Globally*, the entire program execution;
- *Before Q*, before the first occurrence of the state/event *Q*;
- *After Q*, after the first occurrence of the state/event *Q*;
- *Between Q and R*, any part of the program execution between states/events *Q* and *R*;
- *After Q until R*, similar as Between *Q* and *R*, except that the execution continues even if the second state/event never occurs.

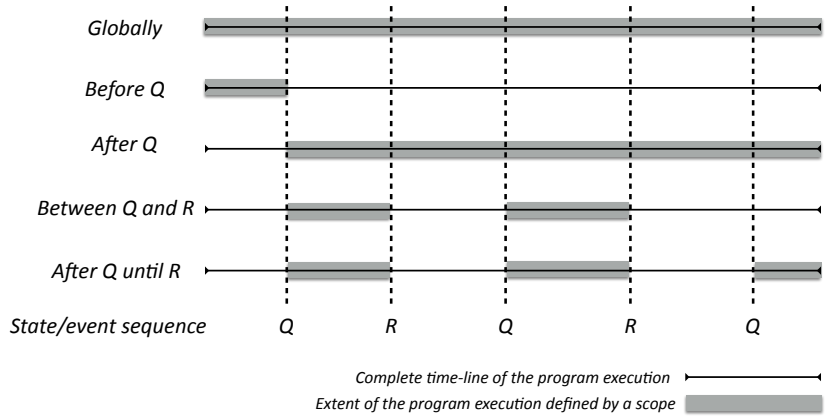


Figure 2.1: Specification pattern scopes as defined by Dwyer et al. [18].

The visualization of the five scopes is given in Figure 2.1. The complete time-line of the program execution is represented by a horizontal line segment. For simplicity, the figure shows the definition of the scopes based on the occurrence of only two different events/states, namely Q and R, which are denoted with vertical dashed lines. The extent of program execution that is captured by a particular scope is denoted by one or more gray rectangles.

One of the limitations of the SPS catalog provided by Dwyer et al. [18] is that it does not contain patterns for specification of real-time properties. For that purpose, Konrad and Cheng introduced a new category of patterns, called *real-time*, suitable for specification of real-time systems. Consequently, the extended catalog of specification patterns is called *real-time specification pattern system* (RTSPS). In the same work, Konrad and Cheng additionally propose a controlled natural language (CNL) representation on top of the formal notations to increase readability and accessibility of specifications to different stakeholders. For illustration, let us consider the following pattern, which captures the bounded response behavior with global scope:

***Globally**, it is always the case that if* P *holds, then* S *holds after at most* t *seconds.*

The literal terminals in the pattern are given in *italic* font and they represent the static part of the pattern, that is, the terminals that are fixed apriori and cannot be changed by the user. The non-literal terminals, given as regular text

in the above pattern can be either Boolean expressions that describe system properties (denoted as P and S for in the illustrative example), or integer values that capture timing aspects (denoted as t in the same pattern). The text in bold denotes the scope of validity of the specified behavior, which is also part of the user-defined input. Given that the semantics of both the scope (Globally) and the behavior (bounded response) are formally defined [12, 18], we can automatically generate the following temporal formula for the given pattern expressed in TCTL: $AG(P \Rightarrow AF_{\leq t}S)$.

As a necessary aid for applying the specification patterns on realistic systems, there exists plethora of academic tools [30–34].

2.3 Sanity Checking of System Specifications

Once the system specifications have been formally encoded, one can apply formal analysis and verification techniques in order to reason about their quality, either with respect to the system model or to establish some properties of the specification in isolation.

The term *sanity checking* has been introduced by Kupferman [35] to denote the process of automatically establishing the quality of formal system specifications represented as a set of temporal formulas with respect to formally defined criteria. In the literature, there is a number of sanity checking approaches that use different sets of formal criteria to assess the quality of the system specifications [35–39], but in this thesis we focus on *consistency*, which means the lack of logically contradicting formulas within a specification. Our aim is to ensure that a set of formally encoded system properties are consistent, that is, free of logical contractions.

The type of sanity checking that we are interested in, namely, assessing the internal consistency of an embedded system specification, does not require a structural or functional model of the system, hence the name *model-free* sanity checking [40]. The benefits of the model-free sanity checking is the possibility to detect errors in the specifications in the early phases of development, thus preventing their propagation into the subsequent artifacts.

Most of the existing sanity checking approaches define sanity checking criteria in such a way that it can be automatically checked using model checking. Despite the benefit of exhaustive sanity checking, such approaches can suffer from number of limitations such as the state-space explosion, or a very long analysis time in cases of complex specifications. Therefore, for early assessments of the specifications' quality, complementary techniques such as

SAT/SMT-based analysis could be beneficial.

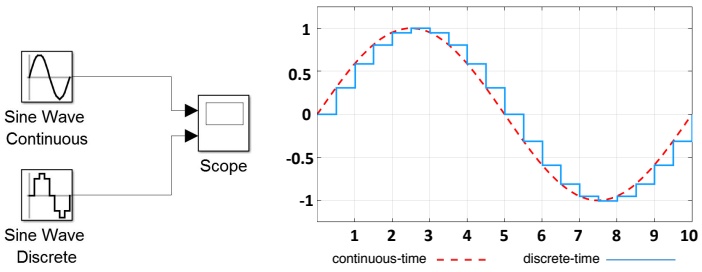
2.4 MATLAB Simulink

MATLAB *Simulink* (commonly referred to as *Simulink*) is a graphical programming environment for modeling and simulation for multi-domain dynamic systems. It represents an extension of the MATLAB environment developed by MathWorks [15]. A Simulink *model* (or Simulink diagram) is a hierarchical representation of a system that is composed of blocks that communicate via signals. Due to its versatility, Simulink has become the *de facto* standard for MBD in many domains, such as the automotive, railways and avionics domain.

Simulink provides a standard library that contains a number of different types of *atomic* and *composite* blocks. An atomic Simulink block represents a fundamental modeling unit that models an input-output relationship or another modeling concept in order to produce an output, either continuously[1] or at specific time points. All of the atomic blocks from the Simulink library have a predefined input-output function, which can be customized via the block specific parameters. For instance, the *Gain* Simulink block performs a multiplication of the value of an input signal by a predefined gain value (scalar or vector). While the input-output function of the *Gain* block cannot be changed, the gain value is user defined and can be changed at both design and run time. The standard library of blocks as provided by Simulink is not complete. In order to facilitate the extension of the standard library of atomic blocks, Simulink introduces the concept of *s-function*, which allows one to define an atomic Simulink block by specifying its input-output function in one of the following programming languages: MATLAB, C, C++, or Fortran. One can additionally apply the concept of masking by using a special extension called *Mask*, which defines the interface of the newly-introduced block and encapsulates its behavior as a black box.

The composite Simulink blocks are used for modeling the hierarchical structure of a Simulink diagram. The most commonly used composite block is the *Subsystem*, which has multiple variations such as *Triggered Subsystem*, *Referenced Subsystem*, etc. Unlike the atomic ones, the composite Simulink blocks do not have a predefined input-output functions. Instead, the computation of the outputs for the composite blocks is modeled through a set of atomic

[1] These blocks are not continuous per se, instead they model the continuous-time behavior in numerical simulations.



(a) (b)

Figure 2.2: Sine-Wave Block: (a) Simulink Diagram and (b) Simulation Result.

blocks. The hierarchical structure of a model is achieved though nesting the composite blocks. The execution semantics of the composite blocks can be either *virtual* or *atomic*. The virtual composite blocks are used to improve the readability of the model and as such do not influence the execution order of the blocks inside the model, whereas the composite blocks that are executed as atomic units enforce Simulink to treat the set of atomic blocks within their structure as an atomic unit of execution. The composite blocks with atomic execution semantics can be conditionally executed based on an external triggering, function call, or enabling input. To facilitate reuse, Simulink allows the contents of a given subsystem to be saved into a separate model file, commonly referred to as *library*.

A Simulink simulation represents a sequential evocation (execution) of the blocks inside the model for a finite time interval. The engine that produces the simulations is called solver [41]. Depending on the nature of a Simulink model, a different solver is used to generate the simulations. The sequential order at which the blocks are executed during simulation is called sorted order or *slist*. Each block has a unique execution order number. The notion of time of a Simulink simulation is modeled via different types of simulation steps. In our work, we assume a fixed-step solver, meaning that the size of the simulation step remains fixed during the entire course of the simulation, and the following two simulation steps: *major* and *minor*. The minor simulation step represents the *fundamental time quanta* that is also an integer fraction of the major simulation step and is used to improve the accuracy of the numerical computation.

Based on how they update their output during simulation, all blocks, be they composite or atomic can be classified into two categories: *continuous-time* and *discrete-time* blocks. The continuous-time blocks produce new outputs at each minor step, whereas the discrete-time blocks produce new outputs at predefined points in time, determined based on the *sampling time* of the block. Another exclusive feature of the discrete-time Simulink blocks is the possibility to delay the first execution, specified through the *offset* parameter of the block. Both the sample time and the offset parameters are expressed as an integer number of major simulation steps. In case the offset of a given block is greater than zero, the subsequent outputs are produced relative to the offset period and not to the beginning of the simulation. Figure 2.2 shows an example of Sine block as modeled in Simulink (Figure 2.2a) and simulation traces for both continuous-time and discrete-time behavior of the block (Figure 2.2b).

The capabilities of Simulink are extended by two supportive tools: *Simulink Design Verifier* (SLDV) [17] and *Simulink Coder* [16], both provided by MathWorks. As advertised by the vendor, the SLDV tool uses formal methods to detect design errors, such as integer overflow, division by zero, dead logic, array access violations, etc. The tool can also verify system requirements expressed as verification objectives, which are in fact simple Simulink models. The Simulink Coder is used for automatic generation of executable C or C++ code.

In this thesis, we focus on two use cases of industrial Simulink models, namely Brake-by-Wire (BBW) and Adjustable Speed Limiter (ASL), both implemented and provided by our industrial partner Volvo Group Trucks Technology (VGTT).

2.5 Formal Modeling and Verification

Formal verification is a set of techniques based on mathematics used to rigorously prove the correctness of a system model expressed in a formal notation. Compared to other verification techniques such as simulation and testing, formal verification techniques are deemed to deliver a higher degree of assurance. Due to this, formal verification techniques can be used for proving the absence of certain types of errors. The formal verification techniques in principle can be divided into two categories: i) *deductive*, used for proving the correctness of the system based on a number of axioms and a set of proof rules, and ii) *algorithmic* or *model checking*, based on techniques that perform systematic and exhaustive exploration of the model's state space in order to determine whether

the system model meets a set of defined logical properties.

The practical difference between the two classes of formal verification techniques boils down to the degree of automation of the verification procedure. While the model-checking-based verification procedures are fully automated, the deductive verification techniques require considerable user interaction during proof construction. Consequently, the model-checking-based formal verification techniques are considered much more suitable for formal verification in industrial settings. In some instances, the model-checking problem can be reduced to a satisfilability problem [42]. In this thesis, we use Satisfilability Modulo Theories [21] for consistency analysis of formalized requirements, and for the bounded model checking of Simulink models.

2.5.1 Satisfiability Modulo Theories and Z3

The problem of determining whether a set of one or more formulas expressing constraints has a solution is called *constraint satisfiability* problem. The most well-known constraint satisfiability problem is the *propositional satisfaction SAT*, where the problem is to decide if a formula over Boolean variables formed using logical connectives can be made true by choosing false/true values of the constituent variables.

To express our constraints, in this thesis we use *first-order logic* (FOL) formulas. A FOL formula is a logical formula formed using logical connectives, variables, quantifiers and function and predicate symbols. A solution for a set of FOL formulas is a *model*, which is an interpretation of all variable, function and predicate symbols that make the given formula true. In addition to the FOL constructs, the formulas that we use contain arithmetic operators such as: $\{<, \leq, =, +, -, *, \div\}$. For checking satisfiability of such formulas that contain symbols that are interpreted by some background theory (such as the theory of arithmetic) we use Satisfiability Modulo Theories (SMT) [21].

The decidability for the SAT problem is NP complete [43], which means that in general case, that is, for all possible interpretations the SAT problem is undecidable, thus it is not feasible to develop a procedure that can solve an arbitrary SAT/SMT problem. To make SMT solving practically possible, most of the decision state-of-art procedures focus on realistic examples and provide means for efficiently solving problems that occur in practice. The basic assumption for such procedures is that the satisfaction of the formulas produced by the verification and analysis tools is due to a small fraction of the formula, while the rest can be deemed irrelevant. In recent years, thanks to the advances in the core algorithms, and the optimizations of data structures and heuristics,

there is tremendous progress in problems that can be solved using SAT/SMT procedures. In addition, a significant role in the advancement is played by the increasingly mature state-of-art tools.

In our work, we use the Z3 [44] SMT solver and theorem prover developed and maintained by the Microsoft RiSE group. The advantage of Z3 is that it has a stable group of developers that maintains the tool, as well as a broad academic community that is actively using it. The input of the tool is a set of *assertions* that can be either declarations or constraints over variables. Originally, the assertions are specified using the SMT-LIB language [45]. For practical reasons, Z3 also provides a number of application programmable interfaces (APIs) for specifying assertions using common programming languages such as C#, Python, Java, etc.

The set of formulas whose satisfiability is to be checked are stored on the Z3 internal stack. The command *assert* adds a new formula to the stack. Once the set of formulas of interest have been placed on the internal stack, the SMT decision procedure is initiated by executing the command *check-sat*, which checks whether there is a solution for the conjunction of all the assertions on the stack. If the set of assertions is satisfiable, Z3 returns the result *SAT*, which can be accompanied by the model that contains the witness assignment of the variables. The model is generated using the command *get-model*. In the opposite case, that is, when the set of assertions on the stack is not satisfied, the tool returns *UNSAT*. Additionally, the *UNSAT* can be augmented with a *minimal* set of inconsistent assertions, which can be generated using the *unsat-core* command.

2.5.2 Model Checking

Model checking is a fully automated verification technique that performs a systematic and exhaustive exploration of the reachable state-space of a model, to prove whether it satisfies a given property modeled in logic [46]. The model-checking procedure is automatically performed by a verifier tool called *model checker*.

The core of model checking is the verification algorithm performed by the model checker. The input of a model checker is a system model expressed in a formal notation and a set of formally specified logical properties. There are two possible outcomes of the model-checking procedure. In case when the model satisfies a given property, the model checker returns a positive answer, which for reachability and some liveness properties (e.g., something good will eventually happen) comes as a witness trace. In case of encountering violation

of a safety property (a bad thing never happens), the model checker generates a counter example, which is usually a path (error trace) to the state that violates the safety property.

Due to its systematic approach and the exhaustiveness of the state-space exploration, the model checking procedure can handle models with state spaces up to a certain size, above which there is not enough memory to store new states. This is known as the *state-space explosion* problem. The state-of-the-art model checking tools, such as UPPAAL [47], Spin [48] or NuSVM [49] use optimal data structures and smart algorithms, thus can be applied on system models with state spaces up to 10^{476} states [50].

Despite the drastic improvements in memory efficiency, performing an exhaustive state-space exploration on industrial models is likely to result in a state-space explosion due to the complexity and high intricacy of the industrial models. In order to avoid this problem for the verification of industrial systems modeled in Simulink, in this thesis we resort to special types of model checking, by which we either compute the probability that a model satisfies a given property using statistical model checking [23] or we exhaustively explore a subset of the complete reachable state-space of the model using a technique called bounded model checking [42]. In the following, we present details about both approaches.

2.5.3 Statistical Model Checking

In this section, we present a more detailed overview of the *statistical model checking* (SMC) by presenting the modeling formalism in which the models can be expressed and the logic that is used to encode the properties of the model.

The main objective of SMC is to compute the probability that a given property is satisfied by a model, based on a finite number of model simulations of finite length [23]. A high-level overview of SMC is given in Figure 2.3. SMC uses series of simulation-based techniques to answer two types of questions:

- *Qualitative*: is the probability of a given property being satisfied by a random system execution greater or equal than some threshold?

- *Quantitative:* what is the probability that a random system execution satisfies a given property?

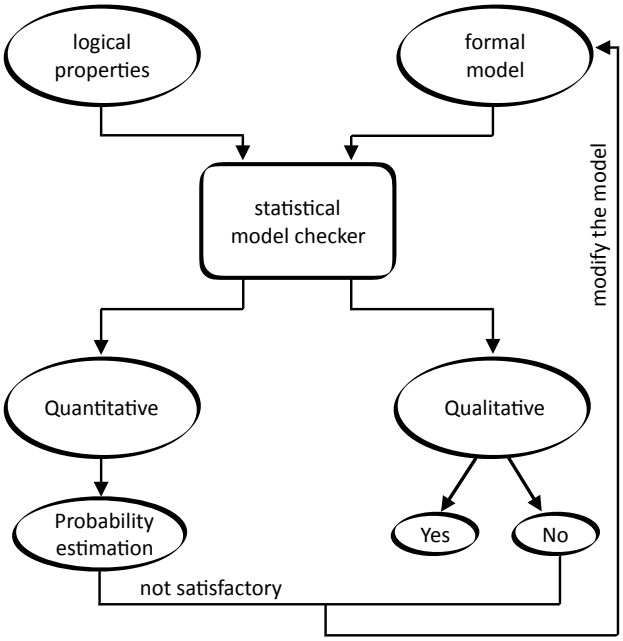


Figure 2.3: Statistical model checking procedure.

The qualitative properties are also referred to as *hypothesis testing*, while the quantitative are called *probability estimation*. In both cases, the answer provided by the procedure will be correct up to a certain level of confidence. Since SMC is less memory intensive than the traditional model checking, it can be used to statistically verify models with infinite state spaces.

In our work, we use UPPAAL Statistical Model Checker (UPPAAL SMC) [24] in our approach proposed to formally analyze semantically transformed Simulink models, which we apply on the use cases provided by our industrial partners. The input of the UPPAAL SMC tool is a network of stochastic timed automata and a set of properties formalized in temporal logic. In the following, we give a brief overview of the timed automata and stochastic timed automata frameworks, as well as the temporal logic used for specifying system properties in UPPAAL SMC.

Timed Automata

Timed automata (TA) [51] represents an extension of finite-state automata with a set of real-valued variables called *clocks*, suitable for modeling the behavior of real-time systems. The clocks are non-negative variables that grow at a fixed rate to denote the passage of time. The only assignment operation of clocks is the reset operation, which sets the clock to zero. The formal definition of a timed automaton (TA) is given as the following tuple:

$$TA = \langle L, l_0, X, \Sigma, E, I \rangle \tag{2.1}$$

where: L is a finite set of locations, $l_0 \in L$ is the initial location, $X \subseteq \mathbb{R}^n$ is a finite set of n clocks, Σ is a finite set of actions, including synchronization and internal actions, $E \subseteq L \times B(X) \times A \times 2^X \times L$ is a finite set of edges of type $e = (l, g, a, r, l')$, where l and l' are the source and the sink locations of the edge, respectively, g is a predicate on $\mathbb{R}^X$ called guard, action label $a \in \Sigma$, and r is the set of clocks that are reset when the edge is traversed. $I : L \to B(X)$ is a function that assigns invariants to locations, which bound the time allowed in a particular location. An edge is going to be traversed if its guard g evaluates to true. $B(X)$ represents the set of formulas called *clock constrains* of the following form $x \bowtie c$, where $x \in X$, $c \in \mathbb{N}$ and $\bowtie \ \in \{<, \leq, =, \geq, >\}$. A clock constraint is downwards closed if $\bowtie \ \in \{<, \leq, =\}$.

The operational semantics of a TA are defined over a *timed transition system* $(S, \to)$, where S is set of states, and the $\to$ is the transition relation that defines how the system evolves by moving from one state to another. A state in the system is represented as a pair (l, v), where l is the location and the v is the valuation of the clocks. A TA evolves by performing a transition, which can be either a *discrete* or a *delay*. By executing a *discrete* transition the automaton instantaneously transitions from one location into another without any time delay, whereas by executing a *delay* transition the automaton stays in the same location while allowing time to pass, which is represented by increasing the value of the clocks. A *path* (or trace) σ of a TA is an infinite sequence $\sigma = s_0 a_0 s_1 a_1 s_2 a_2 ...$ of states alternated by transitions, such that $s_i \xrightarrow{a_i} s_{i+1}$.

The TA modeling framework allows for the systems to be modeled as a set of communicating components. Let $A_1, A_2, ...A_n$ be a set of TA, such that each of them corresponds to an individual component of the system. A *network* of TA (NTA) is simply a parallel composition $A_1 \| A_2 \| \cdots \| A_n$ of a finite number of timed automata, where $\|$ denotes the parallel composition operator.

Stochastic Timed Automata

The stochastic timed automata (STA) [24, 52] is an extension of TA with a delay density function (μ) that represents the set of all density delay functions $\mu_s \in L \times \mathbb{R}^X$, which can be either uniform or exponential distribution, and (γ), which is the set of all output probability functions (γ_s) over the Σ_o output edges of the automaton. The formal definition of a STA is given by the following tuple:

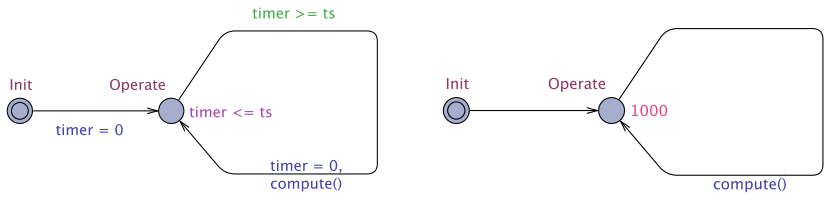
$$STA = \langle TA, \mu, \gamma \rangle \tag{2.2}$$

The stochastic semantics of a timed automaton STA with a corresponding set of states S is defined based on the probability distributions for both delays and outputs for each state $s = (l, v) \in S$ of the automaton [52]. The *delay density function* (μ_s) over delays in $\mathbb{R}_{\geq 0}$, is either a uniform or an exponential distribution depending on the invariant in l. With E_l we denote the disjunction of guards g such that $e = (l, g, o, -, -) \in E$ for some output o. Then, $d(l, v)$ denotes the infimum delay before the output is enabled $d(l, v) = \inf\{d \in \mathbb{R}_{\geq 0} : v + d \models E_l\}$, whereas $D(l, v) = \sup\{d \in \mathbb{R}_{\geq 0} : v + d \models I(l)\}$ is the supremum delay. If the supremum delay $D(l, v) < \infty$, then the delay density function μ_s in a given state s is a uniform distribution over the interval $[d(l, v), D(l, v)]$. Otherwise, when the upper bound on the delays out of s does not exist, then μ_s is an exponential distribution with a rate $P(l)$, where $P : L \to \mathbb{R}_{\geq 0}$ is an additional distribution rate specified for the automaton. The *output probability function* γ_s for every state $s = (l, v) \in S$ is the uniform distribution over the set $\{o : (l, g, o, -, -) \in E \wedge v \models g\}$.

The stochastic semantics of a network of STA (NSTA) subsumes independence between the components [52]. Each component, based on the delay density function and the output probability function repeatedly decides on which output to generate and at which point in time. Consequently, the output will be determined by the component that has chosen to produce output after the minimum delay.

2.5.4 UPPAAL Statistical Model Checker

UPPAAL [47] is an integrated development environment for modeling, simulation and verification of real-time systems, developed as a joint research effort by the Uppsala University and Aalborg University. The tool was initially released in 1995 and since has been constantly updated with new features. UPPAAL Statistical Model Checker (UPPAAL SMC) [53] is an extension of UP-



(a) Timed Automaton (b) Stochastic Timed Automaton

Figure 2.4: UPPAAL timed automaton and stochastic timed automaton.

PAAL model checker for SMC. The input language of the UPPAAL SMC is a NSTA model. In the following section, we present an illustrative example of a stochastic timed automata as modeled in UPPAAL SMC.

UPPAAL SMC Stochastic Timed Automata: Example

In this section we present an illustrative example of an ordinary timed automaton and a stochastic timed automaton as supported by UPPAAL SMC tool.

The input language of the UPPAAL model checker extends the original TA framework with a number of features, including: constants, global and local data variables (integer variables with bounded domain), arithmetic operators, arrays, synchronization channels, urgent and committed locations, as well as definition of procedures using a subset of the C programming language [54]. The input into the UPPAAL SMC tool is a network of stochastic timed automata (NSTA). The NTA in UPPAAL is a parallel composition of a finite set of timed automata over X and Σ, synchronizing over channels and using shared variables.

Figures 2.4a and 2.4b show an example of TA and STA, respectively, as supported by UPPAAL SMC. The automaton in Figure 2.4a shows an UPPAAL TA that models the behavior of a component in the system that periodically executes a predefined computational routine (*compute()*) that maps inputs to outputs. It is composed of two locations: *Init* and *Operate*, out of which *Init* is the initial one, which is denoted by two concentric circles. By taking the edge from *Init* to the *Operate* location the automaton executes an *update* action, which in this particular example resets the clock variable *timer*. The *Operate* location is decorated with an invariant $timer \leq ts$, denoting that the automaton

is allowed to stay in that location and perform delay transitions as long as the value of the clock variable is smaller or equal to the value of the sample time (*ts*). The *Operate* location represents the operational mode of the automaton and has a single looping edge decorated with a *guard* expression *timer* $\geq$ *ts*. The automaton takes the edge on the *Operate* location whenever the guard *timer* $\geq$ *ts* is satisfied, that is, as soon as *timer* == *ts*. When the edge is taken, two update actions are executed: i) the computational routine (*compute()*) that produces output based on the current value of the input, and ii) the reset of the clock variable. The computational routine is encoded as a C function.

In Figure 2.4b we show an example of a STA. The automaton is composed of the same two locations (*Init* and *Operate*) as the TA in Figure 2.4a. This automaton models the behavior of a component that executes continuously, that is, at very small time intervals (similar to a continuous-time Simulink block, Section 2.4). To model an approximation of a continuous behavior of the component, instead of an invariant, we decorate the *Operate* location with a *rate of exponential*. The rate of exponential is in fact a user-defined value for the distribution parameter λ in the delay function that calculates the probability that the automaton leaves the specified location at each simulation step, as follows: $Pr(\text{leaving } \textit{Operate} \text{ after } t) = 1 - e^{-\lambda t}, \lambda = 1000$. Basically, the greater the value of λ, the higher the probability that the automaton leaves the location.

Temporal Logics for Property Specification

In this section, we give an overview of the different temporal logics used in this thesis for specifying properties of timed transition systems.

Computation Tree Logic (CTL) is a branching time logic used for formal specification of finite-state systems [55]. The semantics of CTL is defined over a model M that consists of a non-empty set of states S, a labeling function $Label : S \rightarrow 2^{AP}$ that assigns a set of atomic propositions (AP) to each state in the model, and a successor function $R : S \rightarrow S$ which assigns a set of successor states to each state $s \in S$.

The syntax of a CTL formula consists of quantifiers over paths and path-specific temporal operators. In CTL, there are two path quantifiers: universal, denoted by "A", which reads *"for all paths"*, and existential, denoted by "E", which reads *"there exists a path"*. A valid CTL formula is of the type $\varphi\ U\ \psi$, where "U" ("*until*") represents the basic path-specific temporal operator, that can be combined with either of the path quantifiers. There are two additional derived path-specific temporal operators, namely the $F(Future)$ also denoted by "$\Diamond$", meaning that a formula eventually becomes true ($F\varphi \Leftrightarrow true\ U\ \varphi$),

and the $G(Globally)$ also denoted by "$\Box$", meaning that a given formula is always true ($G\varphi \Leftrightarrow \neg\, F\, \neg\varphi$). There exists also a weaker version of the U operator called *"weak-until"* defined as follows: $\varphi\, W\, \psi \Leftrightarrow (\varphi\, U\, \psi) \vee G\varphi$, which captures all the formulas where the right hand side term (ψ) might never be satisfied.

Timed CTL (TCTL) [56] is an extension of the CTL with real-valued variables called *clocks*. In TCTL each of the path-specific operators has a timed version that uses constraints over clocks. For denoting the timed operators, in this thesis we use the following syntax: $Operator_{\bowtie T}$, where $Operator \in \{U, F, G, W\}$, $\bowtie \in \{<, \leq, =, \geq, >\}$, and T is a numeric bound on the real-valued variable. For instance, the formula $EF_{\leq T}\varphi$ requires that there exists an execution path along which φ eventually becomes true within T time units.

In UPPAAL SMC [52], one can specify probabilistic time-constrained properties in the probabilistic extension of the weighted metric temporal logic (PWMTL). The PWMTL properties of UPPAAL SMC that we use in this thesis are as follows:

$$\psi \;::=\; \mathbb{P}(F_{C\leq c}\varphi) \bowtie p \mid \mathbb{P}(G_{C\leq c}\varphi) \bowtie p \tag{2.3}$$

where C is the observer clock of the automaton under analysis, φ is a state-property with respect to the automaton, $\bowtie \in \{<, \leq, =, \geq, >\}$ and $p \in [0, 1]$.

2.5.5 Bounded Model Checking

Bounded model checking (BMC) [42] is a specialized model checking technique for verification of system properties over a finite subset of the model's state space that is covered by execution paths of finite length. The finite length of the paths, which is usually denoted by *k* is called *bound*. Initially proposed as an efficient refutation technique, it has been shown that BMC can also be used for full verification of underlying designs [57].

Let M be the system model with the set of states S, $I \subseteq S$ be the set of initial states and a state transition relation T, which is a binary relation on S. We write $T(s, s')$ to indicate that the state s is related to state s' via the transition relation T. Based on this, we define a path in M as follows:

$$path(s_{[0,\cdots,i]}) \triangleq \bigwedge T(s_i, s_{i+1}), \forall i \,.\, 0 \leq i{<}n \tag{2.4}$$

Formula (2.4) represents a finite path in model M. The size of a path is determined by the number of transitions that it contains. Same as for any other

form of model checking, one can check different types of properties over the finite paths. In our work, we are interested in checking *safety requirements*, which can be encoded as *invariance properties* over bounded paths. An invariance property *(P)* is a property that holds in every reachable state in model M, which is formally defined as follows:

$$\forall s_0, \cdots, s_n, \forall i \cdot 0 \leq i \leq n-1 \cdot (I(s_0) \wedge path(s_{[0,\cdots,i]})) \implies P(s_i), \quad (2.5)$$

where $P(s_i)$ is a predicate denoting that a given state s_i satisfies property P. Using this definition, one can check the model with respect to invariance properties in one of the following ways: i) using the *forward reachability* procedure, which starts from the set of initial states $I(s_0)$ and repeatedly applies the transition relation T, while checking whether the each new state satisfies P, or ii) one can resort to a *backward reachability* procedure, which starts from a non-initial state ($s_k \cdot 0 < k \leq n$) in which $P(s_k)$ does not hold, and then show that it is not possible to reach an initial state by applying the inverse transition relation.

In any case, one can prove that the model M satisfies an invariance property P by proving the following conjunction:

$$\forall s_0, \cdots, s_n, \forall i \cdot 0 \leq i \leq n-1 \cdot \neg(I(s_0) \wedge path(s_{[0,\cdots,i]}) \wedge \neg P(s_i)) \quad (2.6)$$

where n denotes the length of the longest path of non-repeating states, defined as follows:

$$\max\{i | \exists s_0, \cdots, s_i \cdot I(s_0) \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{m=j+1}^{i} \cdot s_j \neq s_m\} \quad (2.7)$$

Provided that the transition relation $T(s, s')$ can be expressed as a predicate, it is clear how the reachability problem can be reduced to a Boolean satisfiability problem. In cases when the transition relation T is constrained by a background theory, SMT is applied. This representation of the transition relation and the reachability procedure give the following advantages to BMC over the symbolic model checking based on binary decision diagrams: i) it alleviates the infamous state-space explosion of model checking, and ii) it is very efficient for fast detection of errors in bounded traces up to 100 transitions [42].

The invariance-checking BMC procedure terminates when one of the following two conditions is fulfilled: i) Formula (2.6) cannot be satisfied, or ii)

a predefined number of transitions (denoted as k) of the transition relation has been reached. The termination in the first case is due to the fact that a state that does not satisfy property P is detected, in which case a counter-example is generated. In the second case, all of the states along the generated path satisfy the invariance property P. When the procedure terminates according to the second case, a given property is proven to hold in all states between the initial state and the bound, but not beyond that. This makes the procedure incomplete, as there is no information whether the states reachable beyond the bound k satisfy the property or not. However, this is not the case for all models as there are some designs in which the paths might be infinite, but the set of non-repeating states is finite and all of them are reachable within k. Such designs usually contain a *back loop* in the transition relation, meaning that $T(s_i, s_j)$ represents a transition from some current state s_i to a state s_j that has been previously visited. The minimal path that contains the complete set of non-repeating states is called the *reachability diameter*, and its size is called *completeness threshold (CT)*. In our work, we show the existence of the reachability diameter of finite size to perform complete verification of invariance properties over certain Simulink models. For the models that do not have a reachability diameter of finite size, we perform an incomplete invariance checking.

Chapter 3

Research Methodology

A *research method* represents a concrete way of solving research problems. The logical scheme of the research method that is used to obtain answers to scientific questions is called *scientific method* [58]. A collection of well-established and accepted scientific methods, rules and postulates particular for some research discipline is called *research methodology*. In this chapter, we present an overview of the research methodology used to fulfil the research goals of this thesis.

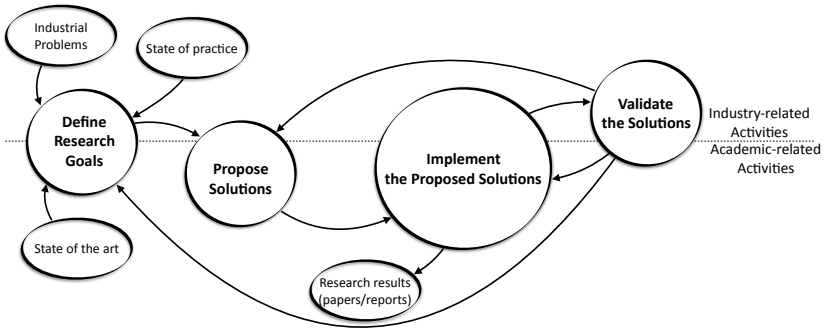


Figure 3.1: The cycle of our research process.

The core of our research methodology is the research process that is illustrated in Figure 3.1. The research process used in this thesis represents an adaptation of the four steps research framework proposed by Holz et al. [59].

Although the original framework is primarily intended for teaching, our adaptation makes it suitable for our research context.

Our research process starts with defining a particular research goal. As the aim of our research is to provide tools and methods that are relevant for the industrial practitioners, all of the research goals are defined based on the following: i) a particular industrial challenge that needs to be addressed, ii) the state of the practice and iii) the state-of-the-art literature. The initial point for defining our research goals are the concrete real-world software engineering challenges that we identify in cooperation with our industrial partners Scania and Volvo Trucks Group Technology (VGTT). Then, we analyze the industrial challenges from a research point-of-view in order to identify the underlying research challenges. A particular challenge might stem from some specific engineering process used by the company or from a set of methods and tools that are currently being used in the development process. In order to formulate a research goal (research objective), we apply the *critical analysis of relevant literature and practice* method [60]. By performing the critical assessment of the relevant literature and practice we make sure that the defined research goal has not been previously addressed in the existing body of knowledge. The formulation of the research goals is not a linear process, as they are iteratively refined and narrowed down until the final version is reached.

During the next step of the research process we propose a solution that addresses an identified research goal. For developing the solutions that address the different research goals defined in this thesis, we use a set of established research methods. In the following, we present the research methods used for developing the contributions in each of the papers that constitute this thesis as listed in Section 1.1. In Paper A [19], we use the *case study research method* [61, 62] in order to assess the expressiveness of an already existing approach for formal system specification in industrial settings. The contribution of the paper is not a method or a tool that addresses a particular research goal, but rather a collection of empirical data that we use to draw conclusions on the expressiveness of an existing method in industrial settings and to identify challenges not addressed in the literature. In contrast to this, in Paper B [22], C [25], and D [26] we propose concrete solutions for the identified research goals through adaptation of already existing approaches for the formal analysis of system specifications encoded as temporal formulas (Paper B) and methods based on existing techniques for the formal analysis and verification of Simulink models, based on statistical model checking (Paper C) and bounded model checking (Paper D). The listed contributions are on two fronts: i) proposing an adaptation of an already existing technique to address

our research goals, which is performed through the *formal proof of correctness research method* [59] that ensures that our adaptation is theoretically and formally sound, and ii) a tool that implements the proposed technique such that the feasibility of the solution can be tested in our context, for which we apply the *proof-of-concept method* [59].

During the last step of the research process, we perform validation of our research results. The main goal of the validation phase is to assess whether our research results are applicable for the earlier identified real-world software engineering challenges in the initial step of the research process. This step is performed in a close cooperation with industry by applying the *proof-by-demonstration* [59] research method, where both researchers and the engineers evaluate the research results. During the validation phase, the following aspects of the results are assessed: i) the scope of the proposed solution, that is, checking whether the proposed solution fully or partially addresses the industrial challenge; ii) scalability, to determine whether the proposed solution can be applied on the actual industrial systems, and iii) usability, that is, does the proposed solution support the transfer of the research results into industrial practice. All of the proposed tools in this thesis are validated using either an operational systems, such as the Fuel Level Display (FLD) system from Scania or the Adjustable Speed Limiter (ASL) from VGTT, or working prototypes, such as the Brake-by-Wire system (BBW) from VGTT. For instance, in Paper E [27], the ProPas tool for the pattern-based formal requirements specification is validated by a group of five engineers from Scania based on a subset of requirements from the FLD system, whereas the BBW Simulink model from VGTT is used for the validation of both the SimPPAAL tool for the formal analysis of Simulink models using statistical model checking (Paper C), and SyMC tool for bounded invariance checking of Simulink models based on the principles of bounded model checking (Paper D). Based on the outcome of the validation phase, we either modify the proposed solution or we modify the initial research goal, which then triggers subsequent changes to adjust the existing solution and the implementation of the tool.

After the validation phase is concluded, the research results composed of the research goal, the solution (both in terms of theoretical advancements and the tool implementation) and the validation results are summarized into a research manuscript, which we then submit for peer review either as a workshop paper, conference paper or journal article. In some instances, when the results are partial but worth sharing with the research community, we publish them as technical reports [63].

Chapter 4

Research Problem

In this chapter, we present the problem (Section 4.1) and the research goals of this thesis (Section 4.2). First, we define the overall research goal based on the actual state of practice and state of the art, which we then decompose into three smaller research goals, such that we can structure our research and map the produced research results to the goals better.

4.1 Problem Statement

One way of enabling practitioners who are not experts in formal methods to create formal system specifications is by providing them with methods and tools that facilitate structured and reusable style of specification that can be expressed in various notations, including a restricted form of natural language, and have precise semantics (a defined relationship between their syntax and the computational model) such that the corresponding formal specifications can be automatically extracted. The specification patterns fulfill these criteria. There is a large body of work on creating different pattern catalogs [11, 12, 18, 64, 65], and a plethora of academic tools that use the specification patterns for creating formal systems' specifications [30–34]. What is missing from the existing endeavors are more studies for generating empirical data regarding the applicability of the specification patterns and their expressiveness for capturing the specifications of industrial systems. Some of the potential questions that await answers are: i) *How can industrial systems' specifications be formalized using the specification patterns, and how many of them?* ii) *Are there types*

of requirements that cannot be captured using the existing set of patterns? iii) *How does an engineer know which pattern to select?* iv) *How to validate that the formalized behavior captures the engineer's intention?*

Once the system specifications have been formally expressed, the next step is to ensure their quality with respect to a certain criterion, such as *consistency*. A consistent system specification is free of internal contradictions, meaning that it is logically consistent. The majority of the existing work on automated formal analysis approaches for assessing the quality of the systems' specifications resort to model checking as an underlying analysis technique [13, 14, 35, 38–40, 66]. For these methods the analysis might suffer from the well-known state-space explosion problem, so its scalability can be potentially limited, especially for large industrial models. Additionally, industry has an imperative need for early checking of correctness of the system specifications for preventing potential specification errors from propagating to subsequent artifacts, including the various system models and ultimately the code as the executable artifact. Consequently, a lightweight and sufficiently scalable formal analysis approach for consistency analysis of systems specifications that can be applied even before a complete behavioral system model exists, might be beneficial.

Most industries, and especially the automotive domain, enjoy the benefits of the MBD paradigm, as models provide a good way of abstracting the engineering problem and documenting the design. Currently, many of the solutions are implemented according to the MBD paradigm using different tools. In the automotive domain, Simulink is the *de facto* standard tool for developing system models. Additional value of the Simulink models is that they can be used as an input into specialized commercial tools for generating code [16], which is later deployed for operation. Assuming that the code generation procedure is correct, one can treat the Simulink models as executable specifications. Under these assumptions, establishing the correctness of the behavioral Simulink models is of utmost importance as it has direct impact on the correctness of the generated code. The formal verification of Simulink models raises challenges on several fronts. First, despite the fact that Simulink is well documented, which includes extensive specification of the atomic and composite blocks, their execution semantics and the input-output functions, there is no formal semantics defined for these models, which can be used to produce a formal counter-part suitable for analysis with the state-of-the-art formal verification tools. Second, considering the high expressiveness of the Simulink models and the complexity and intricacy of industrial systems, even symbolic and exhaustive model checking is likely not to scale, so identifying and em-

ploying alternative verification techniques that are possibly less exhaustive yet rigorous and scalable for industrial application is needed.

4.2 Research Goals Definition

Based on the problem description from Section 4.1, we define the overall research goal (Overall RG) of this thesis as follows:

***Overall RG:* Extend and improve automated and mathematically-rigorous analysis of system requirements specifications and executable design-time Simulink models using state-of-the-art formal verification techniques, towards industrial applicability.**

The overall goal of this thesis is to create means for mathematically-rigorous verification of both system specifications and behavioral models by using state-of-the-art verification methods and tools. One way to facilitate this is to extend existing approaches based on well-established verification methodologies that require less effort and expert knowledge for application in practice, and create tools for automation. The overall goal is defined broadly, and in order to narrow it down and be able to assess the contributions more accurately, we divide it into three smaller goals.

All formal verification techniques rely on properties described in some kind of logical notation (e.g., temporal logic, Boolean logic, etc.). Such properties represent the formal counterpart of the informal system specification. The Achilles heel and "show stopper" of employing formal verification in industry is the daunting task of generating such formal notations. To ease the task, one needs an underlying methodology backed by tool support for the automated generation of formal system specifications. Once the system specification is formally encoded, one can additionally employ formal analysis techniques in order to assure its correctness with respect to consistency, such that all logical contradictions are removed. Hence, we formulate the first research goal (RG1) as follows:

***RG1:* Facilitate the automated generation of formal embedded systems specifications, and their consistency analysis.**

Addressing the first research goal follows two steps: i) investigating the expressiveness of existing reusable approaches (e.g. SPS) for transforming the natural language requirements into formalized requirements when applied on

an industrial use case, and ii) proposing a rigorous method for consistency checking of the formalized requirements, complemented by adequate tool support.

The second challenge that is addressed in this thesis concerns the verification of early design-time executable models. Following the MBD approach, the current industrial practice often resorts to developing executable software models for complex embedded software functions. Among the different toolboxes that enable MBD, MATLAB Simulink is the most popular and the *de facto* standard in many industries, including the automotive industry. The current state-of-practice techniques and tools for verification of such models are restrictive, by either not providing support for the formal verification of high-level system properties, or by failing to scale with the size and complexity of most industrial Simulink models. Consequently, we define the second research goal (RG2) of the thesis as follows:

RG2: Facilitate automated and scalable formal analysis of large Simulink models.

By addressing research goals RG1 and RG2 we establish the foundations of a framework for the formal analysis of requirements, and industrial embedded systems models specified in Simulink. Finally, in order to get more insight about the potential usability of the proposed approaches on industrial systems, we need to investigate their applicability on industrial systems. Consequently, we define the last research goal (RG3) as follows:

RG3: Assess the practical usefulness and scalability of the proposed formal verification approaches on industrial system models.

By addressing RG3, the aim is to gain information on the strengths and limitations of our proposed solutions, when applied on real-world systems, and assess their impact on the quality of industrial embedded systems.

Chapter 5

Thesis Contributions

In this chapter, we give a compact overview of the contributions that address the research goals defined in Section 4.2. The contributions of this thesis are on three main fronts: i) industrially-relevant method and tool for the formal system specification, and an automated consistency analysis approach of formalized system specifications expressed as a set of TCTL formulas, ii) methods and tools for the formal analysis of Simulink models, and iii) assessment of the proposed tools and methods on industrial systems.

5.1 Pattern-based Formal Specification and Automated Consistency Checking of Embedded Systems Requirements

As our first contribution (RC1), we evaluate an existing approach for the pattern-based formal specification of automotive embedded software systems' specifications from Scania, and propose an SMT-based method supported by a tool for checking logical consistency of formally encoded system specifications. This contribution addresses research goal RG1.

In **Paper A** [19], which is also included as **Chapter 8** of this thesis, we reassess the expressiveness of an existing approach called Specification Pattern System [18], more specifically its real-time extension (RTSPS) [12] for formalizing requirements in the automotive domain. Our work has been inspired by an earlier attempt of industrial application of specification patterns by Post

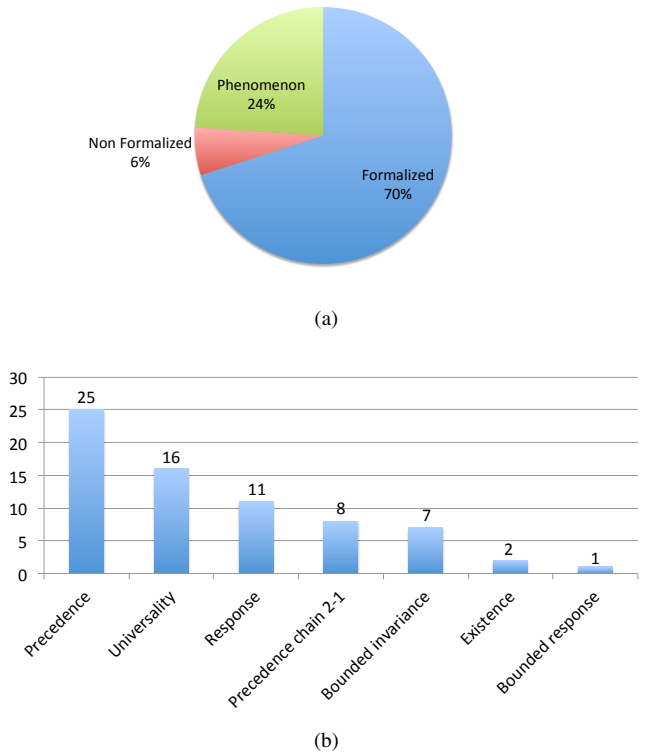


Figure 5.1: Pattern-based formalization results: (a) main formalization results and (b) frequency of the used patterns.

et al. [67], in which the authors show that by using the specification patterns one can capture the behavior of most of the requirements in the automotive domain. Additionally, the same study reports that a small subset of the patterns is usually enough to express most of the requirements. In our work, the goal is to perform a similar study using different systems' specifications from the same domain, with the intention to either strengthen or disprove the claims of Post et al. [67].

The set of requirements selected for formalization in our study consists of 100 requirements that have been extracted from the system specifications of several operational software functions installed in heavy-load vehicles pro-

duced by Scania. The set includes requirements at different levels of abstraction of the system, ranging from high-level requirements up to very detailed implementation specifications. The core part of the study, which is the application of the specification patterns on the set of requirements is performed by a formal methods expert.

The results of our study are given in Figure 5.1. The formalization results given in Figure 5.1a show that 70% of all the requirements can be formalized using the specification patterns; 24% of the remaining requirements belong to the category of so called *phenomenon* requirements that capture the systems configuration and as such can be trivially formalized (no patterns are required). Finally, 6% of the requirements could not be expressed using the specification patterns due to several factors, which include: high complexity, high level of ambiguity, and lack of information. The formalization results from our study are aligned with the results of the previous study in the automotive domain [67].

The second aspect assessed in the study is the frequency of occurrence of the patterns used in the formalization. The goal is to determine the number of the patterns from the RTSPS catalog that have been used in the formalization process and their frequency, that is, how many times each of the patterns has been used. Our results for the pattern frequency are given in Figure 5.1b, and in principle show that a small subset of patterns is enough to formalize most of the requirements. Again, our results are aligned with the previously reported ones [67].

Once the systems' specifications have been formally encoded, one can employ formal analysis techniques to assess their quality with respect to some formally defined criterion. In **Paper B** [22], which is also included as **Chapter 9** of this thesis, we propose an SMT-based consistency analysis approach for system requirements specifications encoded in TCTL. For such system specifications, we propose the following definition for consistency:

Definition 1 (**Inconsistent specification**). *Let* $\Phi = \{\varphi_1, \varphi_2, ..., \varphi_n\}$ *denote the system requirements specification, where each of the formulas* $\varphi_1, \varphi_2, ..., \varphi_n$ *encodes a requirement, respectively. We say that the set* Φ *is inconsistent if the following implication is satisfied:* $\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n \implies False$.

From the definition above, it follows that a system specification is *inconsistent* if there does not exist a truth valuation of the conjunction of all the formulas in the specification. To disprove the inconsistency, following Definition 1, it is enough to provide a witness set of valuations of variables that satisfies the conjunction of all the formulas $\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n$.

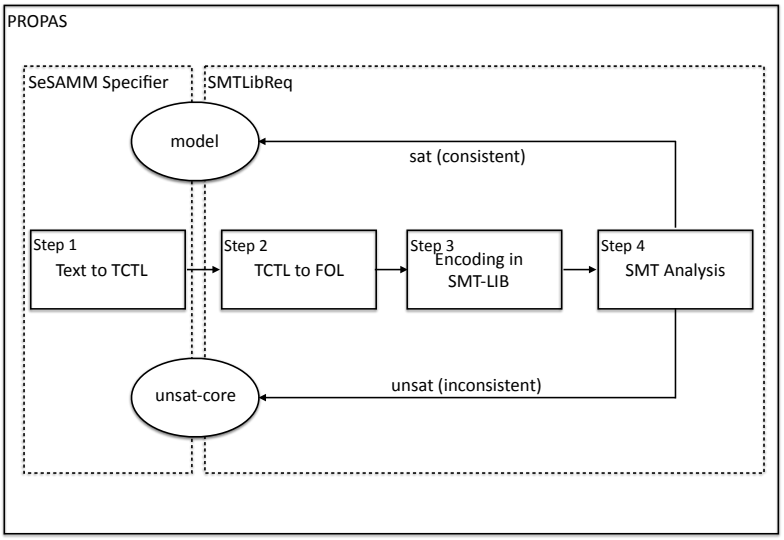


Figure 5.2: Automated SMT-based consistency checking of requirements specifications.

Our methodology for consistency checking of systems specifications is illustrated in Figure 5.2. It consists of four steps given as follows: in *Step 1*, the system specifications expressed in natural language are transformed into a set of TCTL formulas by using the SPS patterns [12, 18]. As the SMT solvers operate over first-order logic (FOL) formulas, in order to bridge the semantic gap between the two formalisms, that is TCTL and FOL, during the *Step 2* the system specification encoded as a set of TCTL formulas is automatically transformed into FOL formulas. The transformation is performed as follows: first the TCTL encoding of the used specification patterns are translated into FOL counterpart by instantiating the semantics of the TCTL path-specific operators and path quantifiers using structured derivations, and second, the non-literal terminals of the patterns (see Section 2.2) that represent the user inputs are mapped into the FOL formulas. During *Step 3*, the FOL formulas are then automatically encoded into SMT-LIB assertions to produce an SMT-LIB script, which can be used as an input to SMT solver. During the same step, the as-

sertions of the generated SMT-LIB script are additionally optimized for solvability by reducing the number of quantifiers and quantified variables using a number of abstraction rules, which are used to simplify the original formulas in order to ensure their analyzability, while preserving the information relevant for detecting potential inconsistencies. Finally, in *Step 4*, we perform the consistency analysis using a state-of-the-art SMT solver (e.g., Z3), which returns the consistency verdict for the considered set of requirements. If the system specification is deemed consistent according to Definition 1 the user gets an affirmative answer by the tool. In the opposite case, that is, when the system specification is not consistent, the negative answer is complemented by a minimal set of logically inconsistent requirements.

The proposed four-step methodology is automated via the PROPAS tool. The tool is created by coupling two major components: our SESAMM SPECIFIER tool [20] that provides means for creating formal system specifications (Step 1 in Figure 5.2), and a library called SMTLIBREQ that completely automates the Steps 2-4 of our consistency analysis methodology given in Figure 5.2.

5.2 Formal Analysis of Simulink Models by Statistical Model Checking

As our second contribution (RC2), we propose an approach for the formal analysis of Simulink models using statistical model checking (SMC). The contribution is given as **Paper C** [25] (which is included as **Chapter 10** of this thesis) and partially addresses research goal RG2. Our approach in essence is a pattern-based, execution-order preserving automatic transformation of atomic and composite Simulink blocks into a NSTA that can then be formally analyzed with UPPAAL SMC. To enable this, first we define the formal syntax and semantics of Simulink blocks and their composition, and second, we show that the transformation is provably correct for discrete-time Simulink models. Our method is supported by the SIMPPAAL tool [25], which we introduce and apply on two industrial Simulink models, a prototype implementation of the Brake-by-Wire (BBW) and the operational Adjustable Speed Limiter (ASL) system. Our work enables the formal analysis of industrial Simulink models by automatically generating STA counterparts. In the following, we present more details of the approach.

We define the syntax of a Simulink block as follows:

Definition 2 (Simulink block). *An atomic Simulink block, denoted by B, is defined as the following tuple:*

$$B = \langle s_n, V_{in}, V_{out}, V_D, \Delta, Init, blockRoutine \rangle \tag{5.1}$$

where: $s_n \in \mathbb{Z}^+$ is the execution order number; V_{in} is a finite set of typed input real-valued variables; V_{out} is a finite set of typed output real-valued variables; is a finite set of typed data (or state) real-valued variables; $\Delta = \{\Delta_0, \Delta_1, \cdots, \Delta_k\}$ represents the totally ordered set of time points at which an output is produced. For discrete-time Simulink blocks, the value of a time point Δ_j is calculated as $\Delta_j = \textit{offset} + j * t_s$, where t_s, $\textit{offset} \in \mathbb{R}_{\geq 0}$ are the sample time and the offset of the atomic Simulink block, respectively, and $0 \leq j \leq k \in \mathbb{N}$ is the index of the element. For continuous-time blocks $\Delta_j = j * t_s$, where t_s is infinitely small. $Init()$ - is an initialization of the data variables; $blockRoutine() = Update(); Output()$ - is the sequential composition of $Output()$ and $Update()$ functions. It captures the functionality of a Simulink block, where: $Output() : V_{in} \times V_D \mapsto V_{out}$ is the output function and $Update() : V_{in} \times V_D \mapsto V_D$ is the update function. A Simulink model is then formally defined as a sequential execution of a set of Simulink blocks represented as follows:

Definition 3 (Simulink model). *A Simulink model is formally defined as a sequential composition of n Simulink blocks that communicate via shared variables, as follows:*

$$S = B_1 \otimes B_2 \otimes B_3 \cdots \otimes B_n, \tag{5.2}$$

where: $s^S = \{s_1, s_2, \ldots, s_n\}$ is an ordered list of execution of blocks $B_1, B_2, \ldots, B_n$, such that $\forall (i,j), i,j \in [1, \cdots, n] \cdot i < j \Rightarrow s_i < s_j$, $V_{in}^S = \bigcup_{i=1}^{n} V_{in}^i$ is the set of input variables of S, $V_{out}^S = \bigcup_{i=1}^{n} V_{out}^i$ is the set of output variables of S, $V_D^S = \bigcup_{i=1}^{n} V_D^i$ is the set of internal state variables, $\Delta_S = \bigcup_{i=1}^{n} \Delta^i$ is the set of time points at which the respective data and output variables are updated, and $(Init; blockRoutine)_S$ is an ordered list of pairs of $(Init, blockRoutine)$, which are executed atomically at given times Δ_i, denoted by $|_{=\Delta_i}$, $1 \leq i \leq n$:

$$\begin{aligned}(Init; blockRoutine)_S \triangleq & (Init_1; blockRoutine_1)|_{=\Delta_1} ; \cdots ; \\ & (Init_i; blockRoutine_i)|_{=\Delta_i}\end{aligned}$$

The operational semantics of a Simulink block is interpreted over a timed transition system T, defined as follows:

$$T = \langle \Sigma, \Sigma_0, L, \rightarrow \rangle \tag{5.3}$$

where: Σ is the set of states, where each state $\sigma = (x|_t, u|_t, y|_t)$ is given by the values $y|_t$ of all output variables y at a given time instance $t \in \mathbb{R}_{\geq 0}$, for given input at time t, that is, $x|_t$, and data at time t, that is, $u|_t$, $\Sigma_0 \subseteq \Sigma$ is the set of initial states, $L = L_a \cup L_t$ represents the set of labels, where $L_a = \{init, blockRoutine\}$ is the set of action labels and $L_t = \{m * \delta, r * \delta\}$ is a set of time labels, and $\rightarrow \subset \Sigma \times L_a \times L_t \times \Sigma$ is the transition relation that consists of the following types of transitions:

$$\sigma_0 \xrightarrow{init, r*\delta} \sigma \iff \begin{cases} \text{if } V_D \neq \emptyset \text{ then } t = t_0 + (r * \delta), \text{ and} \\ \qquad\qquad init() : u = u_0, y_0 = u \\ \text{else } t = t_0 \text{ and } init() : void \end{cases} \tag{5.4}$$

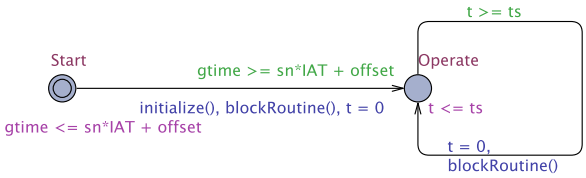
$$\sigma \xrightarrow{blockRoutine, m*\delta} \sigma' \iff t' = t + m * \delta, \text{ and } u' = f(x', u), y' = f(x', u') \tag{5.5}$$

The first transition type called *init* is executed once at the beginning of the blocks' execution only for those that have an internal state, whereas the second type of transitions called *blockRoutine* updates the internal state (if any) and the output variables for given inputs at particular time points denoted as $t + m * \delta$. If in the definition of Δ we instantiate $r = m = 1$ we obtain the continuous-time behavior of blocks that execute the *blockRoutine* infinitely often, that is, at every simulation step δ.

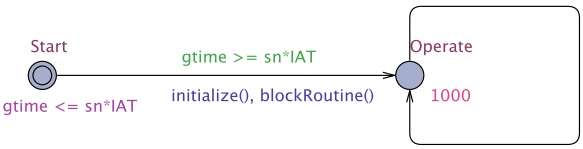
From the above definition, an infinite run ρ of a Simulink block can be defined as the following sequence of states:

$$\rho = \sigma_0 \xrightarrow{init, r*\delta} \sigma_1 \xrightarrow{blockRoutine, m*\delta} \dots \xrightarrow{blockRoutine, m*\delta} \sigma_n \tag{5.6}$$

Further, we use the above semantics of the Simulink blocks given in terms of a timed transition system to transform the Simulink models into a NSTA. The fact that we chose a probabilistic encoding rather than a non-probabilistic one is justified by the scalability of statistical model checking as compared to the traditional one, and this requires the encoding that we propose. The formal definition of an atomic Simulink block given in Definition 2 supports two types



(a)



(b)

Figure 5.3: Templates for transforming Simulink blocks into stochastic timed automata: (a) discrete-time template and (b) continuous-time template.

of computational behavior: i) *continuous* blocks that compute output at each step of the execution, and ii) *discrete* blocks that compute output at specific time points. In order to facilitate the transformation of the atomic Simulink blocks, we propose two (S)TA templates as given in Figure 5.3.

The TA given in Figure 5.3a represents the template for the discrete-time atomic blocks. The automaton is composed of two locations, *Start* and *Operate*, and two edges. The execution modeled by this automaton is as follows: the simulation time is modeled by a global clock, called *gtime*. The automaton is initially in the Start location and stays there until it is scheduled for first execution, which is modeled though the execution order number of the block (*sn*) and the inter-arrival time of the signals within the model (*IAT*). At the first execution, the automaton first initializes the internal state variables by performing the *initialize()* UPPAAL update action, followed by the input-output routine (*blockRoutine()* UPPAAL update action) that calculates the value of the outputs. The last update action is resetting the local clock variable *t*. Both the *initialize()* and *blockRoutine()* are modeled as C functions in UPPAAL. For assuring the correctness of the block routines encoded as C functions, we use the Dafny program verifier [68]. Once the automaton reaches the *Operate* location, it executes the *blockRoutine()* on every sample time interval (*ts*), which is modeled though the loop-edge on the *Operate* location.

The STA in Figure 5.3b represents the template used for modeling the behavior of the continuous-time atomic blocks. The automaton is composed of the same two locations as the discrete-time template and uses the same mechanism for the initial execution. The only difference is in the operational behavior that is modeled. In order to encode the continuous-time behavior of the Simulink blocks, the *Operate* location is decorated with the exponential rate λ, which determines the probability of the automaton leaving the specified location at each simulation step, according to an exponential distribution, formally defined as Pr(leaving Operate after t) $= 1 - \mathrm{e}^{-\lambda t}$.

We provide a proof for the soundness of the transformation of the tuples into STA. The proof covers only Simulink models composed of discrete-time components only. For validation of the Simulink models that contain continuous-time atomic blocks, we resort to comparing the simulation traces between the Simulink model as produced by MATLAB Simulink and the NSTA model obtained in UPPAAL SMC. For the complete proof, we refer our reader to Paper C [25] or to Chapter 10 of this thesis.

Given that in the general case a Simulink model is a hierarchical structure constructed using composite blocks, in the same work we propose a flattening algorithm that is used to transform a hierarchical Simulink model into a flat one. Our flattening procedure serves the following two main purposes: i) it eliminates the Simulink modeling elements that do not contribute to computation, and ii) it assures the preservation of the sequential execution order of the computational blocks as defined in the original Simulink model. After the flattening procedure has been applied, the newly generated flat model is generally of a lower complexity due to the non-computational elements being removed from the model, which include the composite blocks, and virtual modelling atomic elements such as `buss`, `(de)multiplexer`, `goto`, `from`, etc. These elements have the single purpose of improving the readability of the model and do not influence the computation.

Finally, we provide a tool called SIMPPAAL that completely automates the process of generating an analyzable model encoded as a NSTA suitable for analysis using the UPPAAL SMC tool [25], which is an extension of UPPAAL tool for analysis of stochastic hybrid systems. The input to the SIMPPAAL tool is a Simulink model and the block execution order (generated by the MATLAB tool), and as an output it generates a NSTA that is suitable for analysis using the UPPAAL SMC tool. The tool executes all the phases of the transformation process of Simulink model into NSTA in a completely user-transparent way, meaning that it does not require any user intervention during the process.

5.3 Bounded Invariance Checking of Simulink Models

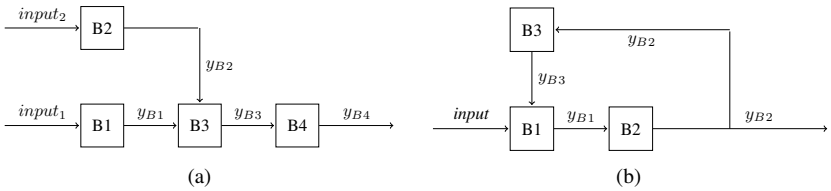


Figure 5.4: Commonly identified Simulink block compositions: (a) Linear composition and (b) Feedback-loop composition.

The third contribution of this thesis (RC3) is an approach and a tool for the formal verification of invariance properties of Simulink models using the principles of bounded model checking (BMC), which completes our contributions towards the research goal RG2. The motivation for investigating this approach is to avoid encoding Simulink model behavior into a stochastic framework, and go beyond probabilistic guarantees of property satisfaction. The proposed approach is presented in **Paper D** [26] (included as **Chapter 11** of this thesis) and is proposed as a complement to our SMC-based analysis approach (see Section 5.2) for Simulink models. In this work, we show how Simulink models can be formally analyzed against invariance properties using BMC. In its basic form, the technique provides means for verification of an underlying model over bounded traces, in a rigorous manner.

Our BMC-based approach unfolds as follows: first, we retain the operational semantics of the Simulink blocks and the flattening procedure as defined for the SMC-based approach (see Section 5.2). This means that the operational semantics are again interpreted over a timed transition system; second, we propose an encoding of the reachable state-space of the Simulink blocks as a sequence of steps each expressed as a constraint, and third, we propose a template-based encoding of the general constraints into assertions encoded into the SMT-LIB format, which can be used as an input in most of the modern SMT solvers. In our work, we use the Z3 [44] SMT solver from Microsoft. Even though BMC procedure is in general incomplete, we show that for certain Simulink designs the verification result is complete. We do this by calculating the bound k and proving that it is indeed a completeness threshold (CT) for

the particular block composition. Consequently, if an invariance property is proven for the reachability diameter then the verification result is complete.

In order to identify recurring Simulink designs, we isolate commonly-used compositions of Simulink blocks from two industrial Simulink models, namely BBW and ASL, both from VGTT, and check whether there exists a CT for such models. Based on the analysis of the Simulink models we identify both the commonly-used blocks and commonly-occurring compositions. The commonly-used blocks belong to three categories, namely: *feedthrough*, *delay* and *s-function*, which when composed together generate the two commonly-occurring compositions: *linear* and *feedthrough* (see Figure 5.4). To be able to reason whether the identified compositions of blocks have CT, we first define how each of the identified block types computes its outputs, respectively, based on the possible transitions for Simulink blocks, which we then use to give formal definitions for the linear and feedthrough compositions of blocks.

Feedthrough (FT) blocks are atomic Simulink blocks characterized by the absence of an internal state and by the immediate execution of the block routine when the input(s) change(s). The type of the block is determined by the type of the input-output function ($f()$ in Formula (5.5) Section 5.2). Based on the possible transitions for atomic Simulink blocks given by Formulas (5.4) and (5.5), the execution trace of a feedthrough block B_{FT}, with $V_{in_{B_{FT}}}$ the set of input variables, $V_{out_{B_{FT}}}$ the set of output variables, and the value of an output variable $y_{B_{FT}} \in V_{out_{B_{FT}}}$ in any state $\sigma_{i_{B_{FT}}}$ of Formula (5.6) (denoted as $y_{i_{B_{FT}}}$), is always defined as follows:

$$y_{i_{B_{FT}}} = f_{B_{FT}}(x_{i_{B_{FT}}}), \tag{5.7}$$

where $x_{i_{B_{FT}}}$ and $y_{i_{B_{FT}}}$ are the valuations of the $x_{B_{FT}} \in V_{in_{B_{FT}}}$ $y_{B_{FT}} \in V_{out_{B_{FT}}}$ in a given state $\sigma_{i_{B_{FT}}}$, respectively.

The execution semantics of a UnitDelay block B_D, which is the only type of delay block present in our use cases, is given as follows: before the first computation, the $init()$ transition is executed, setting $u_{B_D} \in V_{D_{B_D}}$ to the initial value ($initV$) as specified in the block's configuration. After the block starts, the block executes its $blockRoutine()$. If we assume that t denotes the time that has elapsed along the execution of the block (Formula (5.6)), the $blockRoutine()$ is executed whenever $t \; mod \; ts = 0$. Consequently, the value of the output variable y_{B_D} of a UnitDelay block are defined as follows:

$$\begin{cases} u_{i_{B_D}} = initV, \text{ executed before the execution starts} \\ y_{i_{B_D}} = u_{i_{B_D}},\ u_{i_{B_D}} = x_{i_{B_D}}, \text{ if } t \; mod \; ts = 0 \end{cases} \tag{5.8}$$

In our use cases, all of the s-function blocks are either stateless or the internal state is constant, that is, it does not change during execution. Consequently all the s-function blocks can be treated as feedthrough ones.

In order to show the existence of the CT for certain Simulink designs, we propose and prove three theorems for the existence of the CT for the linear and a restricted class of feedback-loop compositions. The first theorem is on the existence of the CT for the linear compositions composed of feedthrough blocks only. In the second theorem, we show that the existence of the CT is still retained for linear compositions composed of feedthrough and delay blocks. The last theorem shows that there is a restricted class of feedback-loop compositions for which there exists a CT. In the proposed theorems, we do not only show that there exists a CT for the specified compositions of blocks, but we also provide means for calculating the size of the reachability diameter for the linear compositions. Our work identifies and shows that Simulink models that contain those certain classes of block compositions that exhibit a CT, can be guaranteed to satisfy an invariance property. For the theorem formulations and the complete proofs we refer our readers to Paper D [26]. Last but not least, we propose the SYMC tool that automates the following steps: i) generation of bounded paths, ii) calculation of CT for the linear-compositions and iii) invariance checking via the Z3 Python API.

5.4 Assessing the practical usefulness and scalability of the proposed approaches on industrial models

As the last contribution (RC4) of this thesis, we present the validation of the proposed approaches and tools for the automated generation of formal systems specifications, and mathematically-rigorous verification of executable Simulink models, which are required to meet the research goal RG3.

In Paper B [22], we validate our SMT-based consistency analysis method by applying the PROPAS tool on a system specification for the Fuel Level Display (FLD) system from Scania. The validation effort in this paper focuses on showing that the approach can be applied on the specification of a realistic system. First, we encode in TCTL the set of 24 requirements using specification patterns, which are then automatically transformed into an SMT-LIB script composed of 36 assertions. The script is then analyzed using the Z3 SMT solver, which proves the system specification of the FLD system consis-

tent in the sense of Definition 1 given in Section 5.2. The proof terminates in seconds on a standard Linux-based laptop. Considering the number of requirements used, generalizing the scalability of the approach is difficult, however, the short analysis time represents a good indicator for the potential usability of the approach. The formal specification of the system has been performed by a formal methods expert, hence that the practical usefulness of the pattern-based specification feature with engineers, is not assessed.

In order to get more insight about the practical usefulness of the pattern-based formal system specification approach, we perform an exploratory case study of the pattern-based approach and the PROPAS tool in industrial setting, which is reported in **Paper E** [27] (included as **Chapter 12** of this thesis). The goal of this paper is to gather empirical data to evaluate the practical usefulness of the pattern-based specification approach and the PROPAS tool in industry. We use the following criteria for the assessment: i) correctness of the application of the specification patterns , and ii) required time for expressing a given system specification using the patterns. The case study involves execution of the following steps: i) select the operational system and the set of requirements to be used in the study, ii) select group of different stakeholders from the embedded software development process in Scania, Sweden, iii) prepare and distribute a comprehensive tutorial on how to use the specification patterns and the PROPAS tool to the selected participants, iv) gathering of qualitative and quantitative data, and finally, v) data analysis and drawing conclusions. Our case study involves the following participants: a software architect for autonomous systems, two development engineers (out of which one previously worked as quality assurance engineer), a requirements coach, and an information/safety architect. The participants in a case study are called *case study subjects* or *CSS* for short. The quantitative data consists of correctness of application of the patterns (expressed as % of correctly specified requirements out of the total set of requirements) and the time (in minutes) for specifying the set of six predefined requirements from the Fuel Level Display (FLD) system, whereas the qualitative data is collected using a survey of ten statements that could be answered using five predefined degrees of satisfaction, ranging from *"completely disagree"* to *"completely agree"*.

The obtained quantitative data shows that in 90% of the cases the correct specification pattern is selected by the engineers (see Table 5.1, where "✓" denotes that a correct pattern is applied for the given requirement and "×" the opposite case), and that on average the engineers spend 26.4 minutes for formalizing the set of six requirements (see Table 5.2). The qualitative data (details provided in Paper E [27] or in Chapter 12) shows that the involved

Table 5.1 Pattern-based requirements specification correctness.

CSS	R_1	R_2	R_3	R_4	R_5	R_6	**total**
CSS_1	✓	✓	✓	✓	✓	✓	100%
CSS_2	×	×	✓	✓	✓	✓	66.67%
CSS_3	✓	✓	✓	✓	✓	✓	100%
CSS_4	✓	✓	✓	✓	✓	✓	100%
CSS_5	✓	✓	✓	✓	✓	×	83.33%
total	80%	80%	100%	100%	100%	80%	90%

Table 5.2 Requirements specification time in minutes.

CSS	R_1	R_2	R_3	R_4	R_5	R_6	**total**
CSS_1	3	2	2	2	4	13	26
CSS_2	3	6	2	2	1	16	30
CSS_3	2	2	2	2	2	16	26
CSS_4	3	2	2	2	2	18	29
CSS_5	2	3	2	2	2	10	21
avg.	2.6	3	2	2	2.2	14.6	26.4

engineers are positively inclined towards the pattern-based system specification through a specialized tool support such as the PROPAS tool. Based on the collected data, we conclude that the pattern-based approach accompanied by an adequate tool support can be practically useful for the engineers. However, in order to assess whether the approach has the potential for a wider industrial adoption a case study of larger scale is required.

We also validate our SMC-based approach for verification of Simulink models implemented in SIMPPAAL on two industrial use cases: Brake-by-Wire (BBW) and Adjustable Speed Limiter (ASL), both from VGTT in Paper C [25]. The validation is carried out by applying our SIMPPAAL tool that automates the process of generating the formal model, on the Simulink diagrams of the above mentioned systems. In order to analyze the scalability of the approach, we applied the SIMPPAAL tool on a prototype Simulink model implementation of the BBW system that contains around 320 Simulink blocks, and on parts of ASL Simulink model that contains more than 4000 blocks. Our preliminary results

Table 5.3 Mapping between the included papers and the research contributions.

	RC1	RC2	RC3	RC4
Paper A	✓			
Paper B	✓			✓
Paper C		✓		✓
Paper D			✓	✓
Paper E	✓			✓

Table 5.4 Mapping between the research contributions and the research goals.

	RG1	RG2	RG3
RC1	✓		
RC2		✓	
RC3		✓	
RC4			✓

show that SIMPPAAL can be successfully applied on Simulink models of the BBW scale; the NSTA model generation terminates within seconds and the resulting model is analyzable. For larger Simulink model sizes, like that of ASL, the tool is applicable up to a certain limit because of technical limitations, such as: i) inability to parse Simulink models that use the concept of library to reference Subsystem blocks, and ii) the limited set of plug-ins for generating the C functions for the blocks input-output functions.

We analyze the NSTA model of the BBW system with respect to six functional properties, out of which three are timed and three untimed. Due to long analysis time, we perform hypothesis testing for one of the properties, whereas for the remaining five we carry out a probability estimation, which in principle is less time consuming. The complete analysis results are given in Paper C [25]. The analysis of the generated model performed by the UPPAAL SMC tool shows encouraging results, however for increased confidence in applicability a more extensive validation is required. The complete process of generating the formal model for analysis is automated and thus transparent for the user, which means that it does not introduce usability difficulties.

The validation of the BMC-based verification approach is presented in Pa-

per D [26]. The approach is validated by assessing the scalability of the SYMC tool that automates the complete process of generating the formal analysis model and performing the bounded invariance checking by means of satisfiability analysis using Z3. Since the model-generation procedure is automated and executed in a "push-button" manner, its practical usefulness is assured by design and implementation. For validation, we apply the SYMC tool on the BBW Simulink model, on which we verify two functional properties. During the analysis we demonstrate both the bounded yet incomplete, as well as the bounded and complete invariance checking. The complete invariance checking is possible for one of the properties, since it specifies the desired behavior of a part of the BBW model that is a linear combination of Simulink blocks, for which the CT exists.

In summary, the mapping between the included papers and the research contributions is given in Table 5.3, whereas the mapping between the research contributions and the research goals is given in Table 5.4.

Chapter 6

Related Work

In this chapter, we present an overview of and comparison to research endeavors related to the three main areas that are considered in our work, including: formal systems specification via specification patterns, automated requirements consistency analysis, and formal analysis of behavioral models of industrial embedded systems specified in Simulink.

Formal system specification. In this thesis we adopt the specification patterns [11, 12, 18] as an engineer-friendly approach for formal system specification suitable for practitioners who are not trained in formal methods. The existing body of work on specification patterns has two major directions: the first one is towards enriching the specification pattern catalog by identifying new patterns, whereas the second one is more focused towards bringing the specification patterns closer to the practitioners. Our work follows the latter research direction.

Most of the existing work in the literature focuses on improving the specification pattern catalog [11,12,18,64,65], with rather few studies that study the applicability of the patterns in real-world scenarios. The main focus our work on the topic is the following: i) performing case studies to test the expressiveness and the practical usefulness of the specification patterns for industrial system specifications, and ii) providing a specialized tool support intended to ease the adoption of the specification patterns in industrial development process.

The suitability, that is, the expressiveness of the specification patterns for formalizing requirements for operational industrial systems has not been addressed in many studies. Post et al. [67] perform a case study in which they

test the expressiveness of the specification patterns for formalizing a set of requirements from the automotive domain. The results of the study show that the specification patterns, in this case the real-time catalog as proposed by Konrad and Cheng [12] is expressive enough for formalizing the selected set of requirements from the automotive domain. There are other studies, in which the expressiveness of the specification patterns is implicitly assessed. For instance, Konad and Cheng [69] successfully apply specification patterns to formalize the specification of an automotive system. The study of Autili et al. [65] focuses on creating a comprehensive catalog of specification patterns by aligning the existent catalogs and introducing new patterns to fill in gaps. For validation, the authors apply the specification patterns on a set of requirements of an ad-hock network protocol. Similarly, Walter et al. [70] use the specification patterns to create formal system specifications, which are then checked for redundancy. In all of these approaches, the complete set of natural language requirements are successfully formalized using specification patterns. In our work [19], we provide empirical data based on our experience of applying the specification patterns on 100 requirements from the automotive domain. Our results are inline with the claims of similar studies [67], and in principle strengthen the earlier claims [67] that specification patterns can be used for creating formal system specifications in the automotive domain.

There is a number of academic tools that have been developed to facilitate the formal requirements specification using the specification patterns. The PROPerty ELucidation system (PROPEL) [30] and Property Specification (Prospect) [32] support requirements specification using disciplined natural language and finite-state automata, relying solely on state-based notations to formally represent requirements behavior. The Property ASSistant (PASS) tool [33] provides means for specification of event-based systems. Other tools such as CHARMY [31] support the design and validation of architectural specifications captured in UML. The PSPWizard tool [65] is very similar to PROPEL and Prospect, yet it exhibits an advantage by providing a more comprehensive catalog of specification patterns.

If compared to the tools listed above, our ProPas tool differs in several ways. First, all previously mentioned tools subsume a specific pattern catalog. In contrast, ProPas does not rely on a predefined catalog of patterns, it rather is designed and developed to be a general tool built on top of the pattern-based approach [20]. Consequently, it has more expressive power than the rest of the tools. Secondly, our ProPas tool provides a mechanism for giving visual feedback to the users. Although not unique with respect to this feature, our tool provides more options for visual feedback compared to existing tools. What is

truly unique about the PROPAS tool is the fact that it has been developed with practitioners in the loop. This has resulted in a tool that engineers can associate themselves with, which can have a positive effect on the adoption of the tool in their everyday work.

Requirements boilerplates is a semi-formal structured specification method that usually lacks formal semantics [71], and it is similar to specification patterns. Farfeleder et al. [72] propose the DODT approach for semi-automatic transformation of natural language requirements into requirements boilerplates. Pang et al. [73] use a combination of requirements boilerplates and domain-specific ontology to generate formal system specifications. Similarly, Mahmud et al. [74] introduce an automotive-specific domain ontology to create formal system specifications expressed in TCTL.

Formal Analysis of Requirements Specification. In our work, we have proposed an approach for automated consistency analysis using Z3 SMT solver. In the following, we list some of the related approaches for automated consistency analysis of requirements specified as temporal formulas.

Barnat et al. [40] propose a model-free sanity checking procedure for consistency analysis of system requirements specification in Linear Temporal Logic (LTL) [75] by means of model checking. The approach has later been extended to support generation of a minimal inconsistent set of requirements [13]. Despite the exhaustiveness, the approach suffers from the inherent complexity of transforming the LTL formulas into automata, especially for complex systems, potentially making it unusable in industrial settings. A similar approach for consistency checking of requirements specified in LTL is proposed by Ellen et al. [76]. The authors introduce a definition for the so-called existential consistency, that is, the existence of at least one system run that satisfies the complete set of requirements. Similar to what we propose, the analysis procedure has been integrated into an industrially relevant tool, aiming at industrial application. The work by Post et al. [39] defines the notion of rt-(in)consistency of real-time requirements. The definition covers cases where the requirements in the system's requirements specification can be inconsistent due to timing constrains. The checking for rt-inconsistency is reduced to model checking, where a deadlock situation implies inconsistency of requirements.

Despite the exhaustiveness of the consistency checking approaches mentioned above, all of them suffer from two major limitations: the time required for generating the requirements model as well as the time for analysis that grows exponentially with the number of requirements that should be analyzed. In comparison, our approach copes well with respect to the time required for

generating the model for analysis. This is due to the fact that the input model for analysis is a set of constraints encoded as Z3 assertions, which can be generated in negligible time as compared to building automata models as required by some of the approaches [13, 39]. Compared to the model-checking-based analysis approaches, the main difference with our method is the exhaustiveness of consistency analysis of the former. Model checking is exhaustive, whereas we sacrifice exhaustiveness for avoiding the potential state-space explosion. The model-checking-based consistency analysis [39, 40] can guarantee the absence of any inconsistencies in the system, while our approach (similarly to [76]) is suitable for checking whether the system specification is realizable as such, that is, if there exists at least one system run that satisfies the conjunction of all the requirements in the specification. A more high-level consistency analysis approach applied at Boolean level, without taking the temporal aspects into consideration is proposed by Mahmud et al. [77]. In cases where inconsistencies are detected, all of the approaches (including ours) are able to generate the minimal set of inconsistent requirements. The above listed characteristics make our approach suitable to be used in the early phases of system requirements specification, where a more lightweight and considerably faster procedure might be more suited.

The quality of the system requirements specifications can be assessed even when they are expressed informally, that is, in natural language. One such approach is proposed by Fabbrini et al. [78, 79], who assesses the quality of the requirements specifications using natural language processing techniques with respect to a quality model. The approach is complemented by an adequate tool support for automating the analysis [80]. A more recent study [81] shows that such techniques can be applied on large sets of industrial requirements to help the engineers prioritize the requirements that are going to be manually analyzed for defects.

Formal Verification of Simulink Models. There is a growing body of work on the formal verification and analysis of Simulink models. In the following, we present some of the state-of-art approaches for analysis of Simulink models, divided in several categories based on how the analysis model is constructed or which type of formal verification technique is employed for analysis.
i) Abstraction of blocks into contracts/theories and their formal analysis. Ferrante et al. [82] use contract-based theory to model the Simulink blocks, and rely on a combination of SAT solvers and the NuSMV model checker for analysis. Hocking et al. [83] use the PVS specification language for writing the specification, and use the PVS theorem prover for analysis. Both steps require

much user interaction, so it is error-prone and requires certain understanding of the formal analysis engines, which is not common among embedded systems engineers, which compared to our approaches that are fully automated represents a disadvantage. In similar fashion, Cavalcanti et al. [84] use the *CircusTime* specification language to construct more realistic analysis models that capture functional, behavioral, and timing aspects of the Simulink models. Similar to our approaches, it is fully automated. Dragomir et al. [85] propose a refinement calculus for reactive systems (RCRS) toolset, which is a fully automated compositional framework for modeling and reasoning about Simulink models. The toolset consists of two main engines, namely *Translator* that transforms a given Simulink model into an RCRS model of the diagram, which is then analyzed via the *Analyzer*. It can be used for static analysis, behavioral type checking and inference for Simulink models. On the down side, the authors do not show how this approach can be used for the verification of Simulink models with respect to properties that describe the functional or behavioral characteristics of the system, which SIMPPAAL focuses on.

ii) Model to model transformation followed by model checking. The approaches based on model-to-model (M2M) transformation aim for minimizing the user intervention during both generation of the formal model and the analysis of the same. The most common strategy is to transform the original Simulink model into some kind of automata framework, which is then analyzed for different properties using model checking.

The approach proposed by Barnat et al. [40] focuses on transforming Simulink models into the language of the LTL explicit model checker DiViNE. The authors show how the latter can be integrated with the Honeywell formal verification environment. The work provides support only for discrete Simulink blocks, yet they show it suitable for the aeronautics industry. Compared to our approaches where we use branching temporal logic, the authors use linear temporal logic for expressing system properties, meaning that the set of properties that can be verified are complementary. Additionally, the approach by Barnat et al. [40] uses explicit LTL model-checking, hence the scalability might be an issue for large Simulink models. Similarly, the approach by Meenakshi et al. [86] proposes a transformation of discrete blocks into a formal model suitable for analysis using the NuSMV model checker. In contrast, Agrawal et al. [87] propose a transformation approach of Simulink models into networks of automata, without providing concrete means for formal verification. The work by Miller [88] proposes a translation from Simulink to Lustre, and enables formal verification with a constellation of model checkers and provers. The transformation of StateFlow design elements

has been addressed in research endeavors by Manamcheri [89] and Jiang et al. [90], in which the authors propose transformation frameworks from StateFlow/Simulink into timed and hybrid automata, respectively, without considering other types of Simulink blocks. Sfyrla et al. [91] propose compositional translation of Simulink models into BIP [92] modeling framework. The transformation procedure is similar to the one that we propose in our SMC-based approach, however, the authors do not report any verification. The formal verification of C-code automatically generated from Simulink modules, as presented by Berger et al. [93], is another example of M2M transformation. The approach has been applied in two case studies of the automotive industry with moderate success because of complexity issues. For the requirements for which finding unbounded correctness proofs is not feasible, the authors advocate the use of subsystem verification. Overall, the paper shows that much human intervention is still required for verifying a Simulink model, with a significant percentage of the time dedicated to requirement formalization, as other studies indicate [19]. Minopoli et al. propose the SL2SX Translator [94] that performs a semi-automated transformation from Simulink diagrams into a hybrid automata formalism, *SpaceEx*, which preserves the model architecture. It is based on a rather simple definition of a Simulink model, which does not account for some of the complex features addressed by SIMPPAAL. It is constrained by the limitations of SpaceEx, but it has been recently complemented with a technique called *syntactic hybridization* [95], which allows analysis of non-linear dynamics.

iii) Generation and abstraction of simulation traces. A third approach, suggested in the PlasmaLab platform [96], consists of generating and collecting simulation traces directly from the Simulink environment, and transforming them, by abstraction, into a state machine representing the system's behavior. PlasmaLab is, in fact, a statistical model checking architecture that can be connected to different simulation engines [96], with Simulink being one of them. It accepts properties specified in bounded linear temporal logic (BLTL) and offers the three basic modes of statistical model checking: simple Monte Carlo, Monte Carlo using Chernoff confidence bound, and sequential hypothesis thesis. The approach has been enhanced with runtime mechanisms for generation of optimal schedules, rare event simulation, and change detection, in order to increase the trust on the properties obtained by SMC [96]. Due to the absence of an input model, the PlasmaLab platform relies on external simulation engines to generate execution traces, and as such is strongly coupled to such tools. Another approach based on statistical model checking is proposed by Zuliani et al. [97]. They use Bayesian statistical model checking for analyzing the

specification, however, the approach has not been applied on Simulink models of industrial systems, and it seems to have practical limitations such as not accepting multi-file Simulink models. In both SIMPPAAL and SYMC the model generation and analysis is completely independent from external tools, as only the system model and the sorted order execution are required once, that is, at the time the formal model is generated. Lastly, PlasmaLab does not require a system model and thus is not hampered by the complexity of M2M transformation. However, for completely discrete-time Simulink models of moderate size and complexity, by using SIMPPAAL one can perform exhaustive model checking thus obtaining either a full guarantee that the property is satisfied, or a counter-example in the opposite case.

Schrammel et al. demonstrate the potential of BMC for verification of industrial systems [98]. In their work, the authors show how an incremental BMC approach can be applied for verification of industrial code. Chaves et al. [99] propose the DSVerifier tool for the formal analysis of digital systems with respect to design errors such as overflow, limit cycle, stability, etc. Again, the analysis is performed over the code, rather than on the Simulink model like in our case. Harde et al. [100] propose a bounded reachability analysis approach for hybrid models using the HySAT tool [101].

Our proposed approaches relies on M2M transformation and statistical model checking or invariance checking based on BMC, but it goes beyond the current state of the art by reducing the modeling effort as M2M transformation is based on templates and fully automated. Additionally, in general, we are supporting a larger number of Simulink blocks (although some of them are still under development), whereas for validation we use real-size industrial examples. We also aim at generating a formal model as close to the Simulink model as possible, so we encode the functions of blocks not as differential equations, when the case, but as C routines that are faithful to the Simulink modeling. In addition, to increase confidence, we also verify the C encoding of the Simulink functions in UPPAAL, by employing the program verifier Dafny [68]. Regarding the BMC-based approach, we show how the completeness of the invariance checking can be established prior to the generation of the analysis model.

Chapter 7

Conclusions and Future Work

In this thesis, we have presented our work on extending and improving the state of the art with respect to methods and tools for the formal specification and analysis of requirements and Simulink models of embedded systems. While conducting our research we have kept industrial applicability and appeal as one of the most important desiderata.

Summary of Contributions. The first contribution of this thesis is an approach for pattern-based formal system specification and its consistency analysis. For the first part of this contribution that concerns the formal encoding of system specifications, we have assessed the expressiveness of the specification patterns (SPS) for capturing industrial system requirements, with particular focus on specifications from the automotive systems domain [19]. Our results show that the specification patterns are expressive enough to capture most of the selected requirements, and that in principle a small set of patterns is sufficient for expressing most of the requirements. Additionally, we complement the pattern-based approach with a specialized tool support called SESAMM SPECIFIER [20], which is not bound to a predefined set of patterns and includes features such as visual feedback on the specified behavior using different forms of visualization. The second aspect of our first contribution is an automated SMT-based consistency analysis approach for system specifications [22], formally encoded as sets of TCTL formulas. Our approach is completely auto-

mated by means of our PROPAS tool. The tool represents an extension of the existing SESAMM SPECIFIER tool, by additionally introducing an SMTLIBREQ that allows one to check the consistency of system specifications, in a completely automated way via the following steps: i) transformation of TCTL formulas into FOL formulas, ii) optimizing the FOL formulas for solving by reducing the number of quantified variables and quantifiers, and their encoding into SMT-LIB format, suitable as input to modern SMT solvers, and iii) consistency analysis using the Z3 SMT solver.

As our next contributions, we propose two approaches for the formal analysis of large Simulink models. Both approaches are based on model checking, which is the most suitable candidate for formal verification of industrial models due to the fully automated verification procedure. Our first approach is based on statistical model checking (SMC), and provides probabilistic guarantees of the correctness of the model with respect to functional and timing requirements, and as such represents a non-exhaustive yet rigorous way of analyzing large Simulink models using the UPPAAL SMC tool [25]. To enable our approach, we propose the following: i) formal definitions of the syntax and semantics of Simulink blocks and their composition in models, ii) a flattening procedure for eliminating the composite blocks from the model structure, and iii) an execution-order-preserving transformation of Simulink blocks into networks of stochastic timed automata, based on two proposed STA templates, one for discrete-time atomic blocks, and one for continuous-time ones. The STA templates encode both the continuous-time and discrete-time execution of the atomic blocks, whereas the input-output function of the atomic blocks is encoded as a C function in UPPAAL SMC. For ensuring that the input-output function is correct, we use the Dafny program verifier [68]. We also provide a proof of soundness for the transformation of completely discrete-time Simulink models. For the models that contain at least one continuous-time atomic block, we resort to simulation-based validation of the transformation. The complete transformation procedure is automated by our SIMPPAAL tool.

The second approach for the formal analysis of Simulink models relies on bounded model checking (BMC) and is suitable for verification of invariance properties [26]. The approach retains the flattening procedure, the template-based transformation of the atomic blocks and the formal semantics of the latter as introduced in our SMC work for Simulink verification [25], with the difference that the model-checking procedure is reduced down to a satisfiability problem. Consequently, we generate an analysis model in a form of an SMT-LIB script in which the reachable state-space is encoded via a number of constraints. In addition, we investigate whether there are some commonly-

occurring design patterns based on commonly-used atomic Simulink blocks in Simulink models, for which the complete reachable state-space can be encoded with a finite number of constraints. For that purpose, we have analyzed two Simulink models, namely Brake-by-Wire (BBW) and Adjustable Speed Limiter (ASL) from VGTT, from which we identified three commonly-used block types (feedthrough, delay and s-function) and two commonly-occurring designs (linear and feedback-loop composition). In order to prove whether the invariance checking for the commonly-occurring designs based on the selected block types is complete, we perform the following: i) give formal definitions for the linear and feedback-loop compositions, ii) propose three theorems that show that for the linear compositions composed of the commonly-used block types the complete reachable-state space is finite, which yields the invariance checking complete, whereas for the feedback-loop compositions in general case the invariance checking is incomplete. The proposed approach is automated by our SYMC tool.

As our last contribution, we perform validation and assess the practical usefulness of the proposed methods and tools. For the validation of the PROPAS tool, we perform consistency analysis of the system specification of the operational FLD system from Scania. The original system specification of the FLD system, which is composed of 24 requirements has been automatically transformed and analyzed in a matter of seconds. In order to assess the practical usefulness of the pattern-based specification approach and the PROPAS tool, we carry out an exploratory case study evaluation with industrial practitioners. For the case study, we have used a selected number of requirements (both functional and timing ones) of the system specification of the operational FLD system and invited a group of five experts in embedded software development in Scania to formalize the requirements in the set, using our PROPAS tool. Based on the collected quantitative data, with respect to the specification correctness and specification time, and the qualitative data based on the personal preferences and opinions of the engineers, we conclude that the pattern-based specification and the PROPAS tool can be practically useful for formally specifying requirements of industrial systems, by engineers lacking expertise in formal methods.

Given the fact that both of the SMC- and BMC-based approaches for formal analysis and verification of Simulink models are completely automated through our SIMPPAAL and SYMC tools, respectively, for validation we apply both of the tools on industrial Simulink models. The capability of the SIMPPAAL tool to transform large industrial Simulink models has been assessed on two instances, namely the BBW and ASL systems from VGTT, which are

composed of around 300 and 4000 blocks, respectively. The transformation procedure finished in matter of seconds. Out of the two generated models, we have analyzed only the BBW model, with respect to 6 properties. The NSTA analysis model generated for the ASL system is not analyzable, due to technical limitations of our tool, which we discuss further in this section. We apply our SYMC tool for bounded invariance checking on the BBW model. The generation procedure of the analysis model terminates within seconds, generating the complete reachable state-space of the model within 200 ms. Given the generated state-space, both the complete and the incomplete invariance checking has been demonstrated.

Scope of Application and Limitations. The main obvious limitation of all of the proposed tools and methods is the restricted validation, which is performed over three systems from two companies from the automotive domain. Even though the proposed approaches are intended to be applied to all kinds of embedded systems, in this thesis, we perform validation only on automotive systems due to the fact that the research presented in this thesis has been carried out in a research project, called VeriSpec[1], which focuses exclusively on the automotive domain.

The pattern-based system specification supported by the PROPAS tool has been applied only on one instance of an operational system. Even though the approach has been assessed by the industrial practitioners and deemed as potentially useful, a more extensive evaluation is required for strengthening our claims. Moreover, the inherent incompleteness of the specification patterns should be taken into consideration. The incompleteness of the specification patterns means that practically there is no finite catalog that contains all the possible patterns for property specification, and in principle it means that there will always be a need that the catalogs are maintained by formal methods experts. Despite the fact that the patterns are expressed in constrained natural language, making them accessible to users not skilled in logics still poses challenges with respect to the accompanying ambiguity related to determining exactly the ordering of events occurrences, specified by the property. A similar validation limitation applies to our automated consistency analysis approach, which has been applied only on one system. For pushing the boundaries of applicability of our consistency-analysis approach, more evaluation involving more systems with potentially larger specifications is required.

Regarding the formal analysis of Simulink models, we report the follow-

[1] http://www.es.mdh.se/projects/343-VeriSpec

ing limitations of our approaches and tools. Our SMC-based formal analysis approach is suitable for analyzing both discrete-time and hybrid Simulink models. However, the proof of correctness of the transformation from the Simulink model into the analyzable STA model targets only the discrete-time models. This is due to the fact that a completely discrete-time model results in a network of timed automata in which the ordered sequence of execution of the components can be guaranteed. In case of hybrid models, due to the presence of continuous-time components and the execution mechanism that relies on a stochastic interpretation, this is not possible. The SIMPPAAL tool on the other hand, exhibits several limitations also, unlike the transformation approach they are of purely technical nature and can be potentially resolved provided enough human resources. Among the limitations that are worth mentioning here, we point out the fact that currently SIMPPAAL cannot parse Simulink models that use the concept of library to reference the contents of subsystem blocks.

Except for the inherent incompleteness problem of the BMC procedure, we are not aware of other theoretical limitations to bounded invariance checking approach. When it comes to the SYMC tool, it exhibits several limitations of technical nature, which should be easy to overcome. The most obvious limitation of the SYMC tool is that currently it accepts as an input a preprocessed Simulink model, with the following characteristics: i) the model is flattened, ii) non-computational block are eliminated from the internal structure such that all connections are between two computational blocks, and iii) it is encoded in JSON[2] format. Additionally, the specification of the invariance properties in the current version of the tool requires training in formal logic and in using the SMT-LIB2 standard [45], which must be addressed in order to make the tool more accessible for a wider audience.

Future Work. There are several directions for future research, in order to improve and advance the work presented in this thesis. The most immediate future work is related to improving the SIMPPAAL tool for generating the formal model. The improvement encompasses mostly technical improvements and adding new features to the tool, such as improvement in the Simulink model parsing feature in order to be able to use the concept of referencing external models as libraries and to remove as many of the external libraries used in the tool that cause unexpected behaviors in the tool, due to limited documentation and immaturity. A similar development can be applied to our SYMC tool, which needs to be improved with respect to parsing of the input Simulink mod-

[2]JavaScript Object Notation (https://www.json.org/)

els. Since parsing Simulink models seems to be the weaker point of our tools, ideally, developing an independent platform that can perform the flattening of the Simulink model and eliminate all the non-computational blocks from its internal structure will solve the problems on both fronts. From a theoretical point of view, we outline the following directions for future research: i) use model-pruning techniques in order to generate more tractable analysis models, ii) explore the possibility of using other modeling frameworks and model-checking tools as compared to what we currently use, iii) identify additional commonly-occurring design patterns and commonly-used Simulink blocks in industrial Simulink models and reason whether for such compositions bounded verification is complete, and iv) investigate whether our bounded invariance checking approach can be extended for verification of invariance properties that include timing constraints.

Concerning the formal system specification using specification patterns, we outline the following directions for future work: i) more empirical studies around formal requirements specification via the specification patterns are necessary in order to identify new gaps and to measure the benefits of using patterns in industry more accurately, and ii) improving the existing PROPAS tool, especially the visual validation mechanism that is intended to help the engineers in visualizing the temporal ordering of the events as described by the pattern.

Bibliography

[1] Martin Hiller. Thoughts on the future of the automotive electronic architecture. In *Presentations at the fuse Final Seminar*, 2016.

[2] Oliver Wyman Group. A comprehensive study on innovation in the automotive industry. http://www.emic-bg.org/files/CarInnovation2015_-engl.pdf, 2015.

[3] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. Software engineering for automotive systems: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 55–71, Washington, DC, USA, 2007. IEEE Computer Society.

[4] Klaus Grimm. Software technology in an automotive company: Major challenges. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 498–503, Washington, DC, USA, 2003. IEEE Computer Society.

[5] UK Ministry of Defence (MoD). *Defence Standard 00-56 (Part 1)/4, Safety Management Requirements for Defence Systems*. Issue 4, UK Ministry of Defence, 2007.

[6] ISO/DIS 26262-1 - Road vehicles Functional safety Part 1 Glossary. Technical report, Geneva, Switzerland, July 2009.

[7] Michael Barr and Anthony Massa. *Programming Embedded Systems*. O'Reilly Media, Inc., 2006.

[8] Michael Barr. Embedded Systems Glossary. http://www.netrino.com/Embedded-Systems/Glossary, Last access: 2019-02-05.

[9] IBM. Rational DOORS - Overview. https://www.ibm.com/us-en/marketplace/rational-doors, Last access: 2019-02-05.

[10] Gursimran Singh Walia and Jeffrey C Carver. A systematic literature review to identify and classify software requirement errors. *Information and Software Technology*, 51(7):1087–1109, 2009.

[11] Matthew B Dwyer, George S Avrunin, and James C Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, pages 411–420. ACM, 1999.

[12] Sascha Konrad and Betty HC Cheng. Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, pages 372–381. ACM, 2005.

[13] Jiří Barnat, Petr Bauch, Nikola Beneš, Luboš Brim, Jan Beran, and Tomáš Kratochvíla. Analyzing Sanity of Requirements for Avionics Systems. *Form. Asp. Comput.*, 28(1):45–63, March 2016.

[14] Mats Heimdahl and Nancy G. Leveson. Completeness and Consistency in Hierarchical State-Based Requirements. *IEEE Trans. Softw. Eng.*, 22(6):363–377, June 1996.

[15] James Dabney and Thomas Harman. *Mastering Simulink*. Pearson/Prentice Hall, 2004.

[16] Mathworks. Simulink Coder - MATLAB Simulink. [Online; Last accessed 2019-01-30].

[17] Mathworks. Simulink Design Verifier - MATLAB Simulink. https://se.mathworks.com/products/sldesignverifier.html. [Online; Last accessed 2019-01-30].

[18] Matthew B Dwyer, George S Avrunin, and James C Corbett. Property specification patterns for finite-state verification. In *Proceedings of the second workshop on Formal methods in software practice*, pages 7–15. ACM, 1998.

[19] Predrag Filipovikj, Mattias Nyberg, and Guillermo Rodriguez-Navas. Reassessing the pattern-based approach for formalizing requirements in the automotive domain. In *RE'14*, pages 444–450, 2014.

[20] Predrag Filipovikj, Trevor Jagerfield, Guillermo Rodriguez-Navas, Mattias Nyberg, and Cristina Seceleanu. Integrating pattern-based formal requirements specification in an industrial tool-chain. In *QUORS*, pages 167–173, 2016.

[21] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. volume 54, pages 69–77, New York, NY, USA, September 2011. ACM.

[22] Predrag Filipovikj, Guillermo Rodriguez-Navas, Mattias Nyberg, and Cristina Seceleanu. Automated smt-based consistency checking of industrial critical requirements. *ACM SIGAPP Applied Computing Review*, 17(4):15–28, 2018.

[23] Håkan LS Younes and Reid G Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification*, pages 223–235. Springer, 2002.

[24] Alexandre David, Dehui Du, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. *arXiv preprint arXiv:1208.3856*, 2012.

[25] Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. SIMPPAAL - a framework for statistical model checking of industrial simulink models. *Submitted to the ACM Transactions on Software Engineering and Methodology*.

[26] Predrag Filipovikj, Guillermo Rodriguez-Navas, and Cristina Seceleanu. Bounded invariance checking of simulink models. In *The 34th ACM/SIGAPP Symposium On Applied Computing*, April 2019.

[27] Predrag Filipovikj and Cristina Seceleanu. Specifying industrial system requirements using specification patterns: A case study of evaluation with practitioners. In *Proceeedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, 2019.

[28] Frederick P. Brooks, Jr. *The Mythical Man-month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[29] Bran Selic. Model-Driven Development: Its Essence and Opportunities. In *Ninth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2006), 24-26 April 2006, Gyeongju, Korea*, pages 313–319, 2006.

[30] Rachel L. Cobleigh, George S. Avrunin, and Lori A. Clarke. User guidance for creating precise and accessible property specifications. In *SIGSOFT '06/FSE-14*, pages 208–218. ACM, 2006.

[31] Paola Inverardi, Henry Muccini, and Patrizio Pelliccione. Charmy: An extensible tool for architectural analysis. In *ESEC/FSE-13*, pages 111–114. ACM, 2005.

[32] Oscar Mondragn, Ann Q. Gates, and Steven Roach. Prospec: Support for elicitation and formal specification of software properties. *Electronic Notes in Theoretical Computer Science*, pages 67 – 88, 2003.

[33] Daniela Remenska, Tim A. C. Willemse, Jeff Templon, Kees Verstoep, and Henri Bal. *Property Specification Made Easy: Harnessing the Power of Model Checking in UML Designs*, pages 17–32. Springer Verlag, 2014.

[34] Rachel L. Smith, George S. Avrunin, Lori A. Clarke, and Leon J. Osterweil. Propel: An approach supporting property elucidation. In *ICSE '02*, pages 11–21. ACM, 2002.

[35] Orna Kupferman. Sanity checks in formal verification. In *CONCUR 2006*, pages 37–51, 2006.

[36] Matthew S. Jaffe, Nancy G. Leveson, Mats P. E. Heimdahl, and Bonnie E. Melhart. Software Requirements Analysis for Real-Time Process-Control Systems. *IEEE Trans. Softw. Eng.*, 17(3):241–258, March 1991.

[37] Mats Heimdahl and Nancy Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering*, 22(6):363–377, 1996.

[38] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated Consistency Checking of Requirements Specifications. *ACM Transactions Software Engineering Methodology*, 5(3):231–261, July 1996.

[39] Amalinda Post, Jochen Hoenicke, and Andreas Podelski. rt-inconsistency: a new property for real-time requirements. In *FASE 2011*, number 6603 in LNCS, pages 34–49. Springer, 2011.

[40] Petr Bauch Jiri Barnat and Lubos Brim. Checking sanity of software requirements. In *SEFM 2012, LNCS 7504*, pages 48–62, 2012.

[41] Mathworks. Choose a solver - MATLAB Simulink. https://se.mathworks.com/help/simulink/ug/types-of-solvers.html. [Online; Last accessed 2019-03-05].

[42] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in computers*, 58(11):117–148, 2003.

[43] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[44] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[45] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.

[46] Joost-Piter Katoen. *Concepts, Algorithms and Tools for Model Checking*. 1999.

[47] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAALin a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.

[48] Gerard J. Holzmann. The Model Checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[49] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, pages 359–364, London, UK, UK, 2002. Springer-Verlag.

[50] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[51] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

[52] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. *Statistical Model Checking for Networks of Priced Timed Automata*, pages 80–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[53] Alexandre David, Kim Larsen, Axel. Legay, M. Mikučionis, and D.B. Poulsen. UPPAAL SMC tutorial. *STTT Journal*, 17(4):397–415, 2015.

[54] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Bertinora, Italy, September 13-18, 2004, Revised Lectures*, pages 200–236, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[55] Edmund Clarke, Allen Emerson, and Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, pages 244–263, 1986.

[56] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, pages 2–34, 1993.

[57] Leonardo De Moura, Harald Rueß, and Maria Sorea. Bounded model checking and induction: From refutation to verification. In *International Conference on Computer Aided Verification*, pages 14–26. Springer, 2003.

[58] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*, pages 126–130, 2002.

[59] Hilary J. Holz, Anne Applin, Bruria Haberman, Donald Joyce, Helen Purchase, and Catherine Reed. Research methods in computing: What are they, and how should we teach them? *SIGCSE Bull.*, 38(4):96–114, June 2006.

[60] Marvin V. Zelkowitz and Dolores Wallace. Experimental Validation In Software Engineering. *Information and Software Technology*, 39:735–743, 1997.

[61] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[62] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *IEEE software*, 12(4):52–62, 1995.

[63] Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Guillermo Rodriguez-Navas, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. Analyzing industrial simulink models by statistical model checking. Technical report, March 2017.

[64] Lars Grunske. Specification patterns for probabilistic quality properties. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 31–40. IEEE, 2008.

[65] Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *Software Engineering, IEEE Transactions*, pages 620–638, 2015.

[66] Amalinda Post, Jochen Hoenicke, and Andreas Podelski. Vacuous real-time requirements. In *RE 11*, pages 153–162. IEEE, 2011.

[67] Amalinda Post, Igor Menzel, Jochen Hoenicke, and Andreas Podelski. Automotive behavioral requirements expressed in a specification pattern system: a case study at bosch. *Requirements Engineering*, 17(1):19–33, 2012.

[68] Rustan Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR'10*, pages 348–370. Springer, 2010.

[69] Sascha Konrad and Betty HC Cheng. Facilitating the construction of specification pattern-based properties. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 329–338. IEEE, 2005.

[70] Benedikt Walter, Jakob Hammes, Marco Piechotta, and Stephan Rudolph. A formalization method to process structured natural language to logic expressions to detect redundant specification and test statements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 263–272. IEEE, 2017.

[71] Ajitha Rajan and Thomas Wahl. *CESAR: Cost-efficient methods and processes for safety-relevant embedded systems*. Number 978-3709113868. Springer, 2013.

[72] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Herbert Zojer, and Christian Panis. Dodt: Increasing requirements formalism using domain ontologies for improved embedded systems development. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 271–274. IEEE, 2011.

[73] Cheng Pang, Antti Pakonen, Igor Buzhinsky, and Valeriy Vyatkin. A study on user-friendly formal specification languages for requirements formalization. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 676–682. IEEE, 2016.

[74] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic. In *International Conference on Software Engineering and Formal Methods*, pages 332–348. Springer, 2017.

[75] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[76] Christian Ellen, Sven Sieverding, and Hardi Hungar. Detecting consistencies and inconsistencies of pattern-based functional requirements. In *FMICS*, volume 8718 of *LNCS*, pages 155–169. Springer, 2014.

[77] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. ReSA Tool: Structured Requirements Specification and SAT-based Consistency-checking. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016.*, pages 1737–1746, 2016.

[78] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. An Automatic Quality Evaluation for Natural Language Requirements. In *in Proceedings of the Seventh International Workshop on RE: Foundation for Software Quality (REFSQ2001*, pages 4–5, 2001.

[79] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the Use of an Automatic Tool. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, SEW '01, pages 97–, Washington, DC, USA, 2001. IEEE Computer Society.

[80] Stefania Gnesi, Fabrizio Fabbrini, Mario Fusani, and Gianluca Trentanni. An automatic tool for the analysis of natural language requirements. *CRL Publishing: Leicester*, 20:53–62, 2005.

[81] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. *Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain*, pages 344–360. Springer International Publishing, Cham, 2017.

[82] Orlando Ferrante, Luca Benvenuti, Leonardo Mangeruca, Christos Sofronis, and Alberto Ferrari. Parallel NuSMV: A NuSMV extension for the verification of complex embedded systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7613 LNCS, pages 409–416, 2012.

[83] Ashlie B. Hocking, M. Anthony Aiello, John C. Knight, and Nikos Aréchiga. Proving Critical Properties of Simulink Models. In *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, volume 2016-March, pages 189–196, 2016.

[84] Ana Cavalcanti, Alexandre Mota, and Jim Woodcock. Simulink timed models for program verification. In *Theories of Programming and Formal Methods*, pages 82–99. Springer, 2013.

[85] Iulia Dragomir, Viorel Preoteasa, and Stavros Tripakis. The refinement calculus of reactive systems toolset. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 201–208. Springer, 2018.

[86] B. Meenakshi, Abhishek Bhatnagar, and Sudeepa Roy. Tool for Translating Simulink Models into Input Language of a Model Checker. In *ICFEM*, pages 606–620. Springer, 2006.

[87] Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic Translation of Simulink/Stateflow Models to Hybrid Automata using Graph Transformations. *ENTC Journal*, 109:43–56, 2004.

[88] Steven P. Miller. Bridging the Gap Between Model-Based Development and Model Checking. In *TACAS*, pages 443–453. Springer, 2009.

[89] K. Manamcheri Sukumar. Translation of Simulink-Stateflow Models to Hybrid Automata. 2011.

[90] Yu Jiang, Yixiao Yang, Han Liu, Hui Kong, Ming Gu, Jiaguang Sun, and Lui Sha. From Stateflow Simulation to Verified Implementation: A Verification Approach and A Real-Time Train Controller Design. In *RTAS'16*, pages 1–11, April 2016.

[91] Vassiliki Sfyrla, Georgios Tsiligiannis, Iris Safaka, Marius Bozga, and Joseph Sifakis. Compositional translation of simulink models into synchronous bip. In *IEEE Fifth International Symposium on Industrial Embedded Systems - SIES 2010, University of Trento, Italy, July 7-9, 2010*, pages 217–220. IEEE, 2010.

[92] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*, pages 3–12. IEEE, 2006.

[93] Philipp Berger, Joost-Pieter Katoen, Erika Ábrahám, Md Tawhid Bin Waez, and Thomas Rambow. Verifying auto-generated c code from simulink. In Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink, editors, *Formal Methods*, pages 312–328, Cham, 2018. Springer International Publishing.

[94] Stefano Minopoli and Goran Frehse. Sl2sx translator: From simulink to spaceex models. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, HSCC '16, pages 93–98, New York, NY, USA, 2016. ACM.

[95] Nikolaos Kekatos, Marcelo Forets, and Goran Frehse. Constructing verification models of nonlinear simulink systems via syntactic hybridization. In *Proceedings of Applied Verification for Continuous and Hybrid Systems*, 2017.

[96] Axel Legay and Louis-Marie Traonouez. Statistical model checking of simulink models with plasma lab. In Cyrille Artho and Peter Csaba Ölveczky, editors, *Formal Techniques for Safety-Critical Systems*, pages 259–264, Cham, 2016. Springer International Publishing.

[97] Paolo Zuliani, André Platzer, and Edmund M Clarke. Bayesian statistical model checking with application to stateflow/simulink verification. *Formal Methods in System Design*, 43(2):338–367, 2013.

[98] Peter Schrammel, Daniel Kroening, Martin Brain, Ruben Martins, Tino Teige, and Tom Bienmüller. Successful use of incremental bmc in the automotive industry. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 62–77. Springer, 2015.

[99] Lennon Chaves, Iury Bessa, Lucas Cordeiro, Daniel Kroening, and Eddie Lima. Verifying digital systems with matlab. In *Proceedings of the 26th ACM SIGSOFT Int. Symposium on Software Testing and Analysis*, pages 388–391. ACM, 2017.

[100] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using hysat. In *IN ICONS '08: Proceedings of the Third International Conference On Systems*, pages 196–201, 2008.

[101] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.